Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Matevž Gačnik

Porazdeljeno procesiranje, analiza in vizualizacija podatkov z mehanizmi visoke skalabilnosti

Magistrsko delo

Ljubljana, 2016

Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Matevž Gačnik

Porazdeljeno procesiranje, analiza in vizualizacija podatkov z mehanizmi visoke skalabilnosti

Magistrsko delo

Mentor: doc. dr. Matija Marolt

Ljubljana, 2016

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Večna pot 113
1000 Ljubljana, Slovenija
telefon: 01 47 98 100
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Številka: 158-MAG-RI/2016
Datum: 06. 04. 2016

Matevž GAČNIK, univ. dipl. inž. rač. in inf.

Ljubljana

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Porazdeljeno procesiranje, analiza in vizualizacija podatkov z mehanizmi visoke skalabilnosti**

Parallel data processing, analysis and visualization using high scalability mechanisms

Tematika naloge:

V nalogi raziščite možnosti za skalabilno, porazdeljeno in uteženo izvajanje velike količine operacij na več procesnih enotah, ki delujejo v oblaku. Prikažite kako je mogoče razviti sisteme za procesiranje, ki imajo minimalne procesne, kot tudi časovne zahteve ob zahtevi po spremembi količine procesne moči. Razložite načine implementacije elastičnega prilagajanja glede na potrebe z uporabo procesiranja v oblaku.
Predlagajte načine za filtriranje uporabnih podatkov po kriterijih, ki so za problemsko domeno pomembni. Razvijte možnosti za napredno filtriranje podatkov glede na zahteve analiz.
Glede na podan model procesiranja predlagajte načine in izvedbo vizualizacije naprednih analiz nad zajetimi in procesiranimi podatki v obliki intuitivnih in interaktivnih grafov, grafikonov, besednih oblakov ali drugih za uporabnika primernih prikazov.

Mentor:

doc. dr. Matija Marolt

Fakulteta za računalništvo in informatiko 2 Univerza v Ljubljani

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu zaključnega dela

Spodaj podpisani Matevž Gačnik, vpisna številka 24950319, avtor zaključnega magistrskega dela z naslovom:

Porazdeljeno procesiranje, analiza in vizualizacija podatkov z mehanizmi visoke skalabilnosti (angl. *Parallel data processing, analysis and visualization using high scalability mechanisms*)

Izjavljam

1. da sem pisno zaključno delo študija izdelal samostojno pod mentorstvom doc. dr. Matije Marolta;
2. da je tiskana oblika pisnega zaključnega dela študija istovetna elektronski obliki pisnega zaključnega dela študija;
3. da sem pridobil vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v pisnem zaključnem delu študija in jih v pisnem zaključnem delu študija jasno označil;
4. da sem pri pripravi pisnega zaključnega dela študija ravnal v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil soglasje etične komisije;
5. soglašam, da se elektronska oblika pisnega zaključnega dela študija uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
6. da na UL neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja pisnega zaključnega dela študija na voljo javnosti na svetovnem spletu preko Repozitorija UL;
7. dovoljujem objavo svojih osebnih podatkov, ki so navedeni v pisnem zaključnem delu študija in tej izjavi, skupaj z objavo pisnega zaključnega dela študija.

V Ljubljani, 9. avgust 2016 Podpis študenta: 

Zahvaljujem se mentorju, doc. dr. Matiji Maroltu, za odzivno in strokovno pomoč ter usmeritve.

Zahvaljujem se moji dragi Petri in mami Tatjani, ker sta vedeli kdaj in kako me podpreti, kdaj prijetno vzpodbujati in tudi kdaj opomniti.

Iskrena hvala vsem.

Mojim puncam.

Kazalo

Kazalo slik

Kazalo tabel

Kazalo algoritmov

Seznam uporabljenih kratic

kratica	**angleško**	**slovensko**
SLA	service level agreement	sporazum o ravni storitev
API	application programming interface	programski vmesnik aplikacije
P2P	peer to peer	enak - enak

Povzetek

Naslov: Porazdeljeno procesiranje, analiza in vizualizacija podatkov z mehanizmi visoke skalabilnosti

V nalogi smo predstavili konceptualni in izvedbeni model za skalabilno, porazdeljeno in uteženo izvajanje velike količine operacij na več procesnih enotah, ki delujejo v oblaku.

Delo prikazuje način razvoja sistemov za procesiranje, ki imajo minimalne procesne, kot tudi časovne omejitve ob zahtevi po spremembi količine procesne moči. Razloženi so načini implementacije elastičnega prilagajanja glede na potrebe z uporabo procesiranja v oblaku.

V delu smo preučili načine za filtriranje uporabnih podatkov po kriterijih, ki so za problemsko domeno pomembni. Prikazane so možnosti za napredno filtriranje podatkov glede na zahteve analiz.

Delo podaja tudi načine in izvedbo vizualizacije naprednih analiz nad zajetimi in procesiranimi podatki v obliki intuitivnih in interaktivnih grafov, grafikonov, besednih oblakov ali drugih za uporabnika primernih prikazih.

Ključne besede: porazdeljeno procesiranje, analiza, paralelizem, oblak, računske operacije, skalabilnost

Abstract

Title: Parallel data processing, analysis and visualization using high scalability mechanisms

In this work we present conceptual and implementation model for scalable, distributed and balanced execution of large number of compute operations running on multiple processing units in the cloud.

We provide system development methods for large scale processing with minimal time constraints and limitations in regard to increasing scale-out parallelism in the cloud. Implementation details regarding elastic adjustment to processing units are discussed in connection to required processing power needed in a cloud environment.

Work provides filtering approaches for useful data in the described problem domain. We present options for advanced data filtering in multiple stages, which correlate with needed analyses requirements.

At the end of this work we present ways of visualization of advanced analysis of gathered data in a form of intuitive and interactive UI components, graphs, word clouds and other user acceptable views.

Keywords: parallel processing, analysis, parallelism, cloud, compute operations, scalability

1 Uvod

Obvladovanje, shranjevanje in procesiranje velike količine podatkov predstavlja v današnjem času potrebo v t.i. big-data rešitvah, ki poskušajo iz ogromnega števila podatkov izluščiti pomembne ali karakteristične informacije.

V pričujočem magistrskem delu predstavljamo generičen model porazdeljenega procesiranja, analize in vizualizacije takih podatkov. Model je generičen v smislu možnosti uvedbe poljubnih vrst podatkov in izvajanja drugačnih operacij procesiranja, analiz in vizualizacij, pri čemer osnova in pristop k modelu ostaja enak.

Količina podatkov iz različnih virov (generične vsebine na internetu, IoT, družbena omrežja, ...), ki imajo potencial za nadaljnjo vsebinsko analizo, se povečuje [1]. Količina vsebin na internetu se povečuje celo eksponentno [2]- podatki, ki so primerno za nadaljnjo analizo posledično prav tako rastejo [3].

Usmerjanje pravih informacij k pravim naslovnikom je pri velikih količinah podatkov problem, ki ga je potrebno reševati na skalabilen način z možnostjo hitrega prilagajanja količine zahtevanih računskih virov. Prilagodljivo porazdeljeno procesiranje poljubnih operacij, ki je hkrati ekonomsko učinkovito, trenutno predstavlja velik izziv. Teoretični pogledi porazdeljevanje bremena v oblaku so delno že raziskani [4] [5] [6], vendar je generalizacija operacij, posebej pa možnost hitrega in učinkovitega skaliranja arbitrarnih operacij z mehanizmi, ki so dostopni večini razvijalcev, še vedno težavna [5]. Trenutne tehnologije procesiranja v oblaku omogočajo velike možnosti povečevanja procesorskih virov, vseeno pa je nepremišljeno povečevanje virov večkrat podvrženo njihovi neučinkoviti uporabi in ne upravičuje zvišanja stroškov.

Primarni cilj magistrskega dela je definicija modela za pridobivanje podatkov iz tokovnih in povpraševalnih izvorov, procesiranje teh podatkovnih tokov, njihova analitika ter grafična vizualizacija, ki je aplikativna v množici uporabniških rešitev. V okviru naloge obravnavamo področje usmerjanja velike količine sporočil od pošiljateljev do naslovnikov. Podrobno preučujemo porazdeljeno obliko mehanizma založnik-naročnik, ki omogoča visoko stopnjo skalabilnosti v primeru širokih in visoko pretočnih podatkovnih tokov, primerna pa je tudi za povpraševalne tokove, ki omogočajo povpraševanje po specifičnih, vnaprej določenih virih.

Model, ki ga v delu raziskujemo, predstavlja osnovo za masovno procesiranje podatkov z načini porazdeljevanja računskega dela, ki se izvaja v oblaku. Uporabljeni pristopi so uporabni pri večini ponudnikov takih storitev in niso specifično vezani na tehnologijo, ponudnika ali problemsko domeno.

V nalogi predstavljamo načine za porazdeljevanje dela med sebi enakimi procesorskimi entitetami (vlogami), ki imajo avtonomno delovanje, vendar vedno izberejo tudi vodjo, ki procesno delo razdeljuje. Naloga se ukvarja z učinkovitimi načini za izbiro vodij v primerih velikega števila procesorjev, ki svoje delo izvajajo na različnih porazdeljenih lokacijah sveta ter načini za zagotavljanje vsaj enega delujočega vodje. Hkrati so podrobno opredeljeni načini za učinkovito zagotavljanje idempotentnosti izvajanja operacij, kar ima v visoko latentnem okolju izvajanja procesiranja v oblaku pomembno vlogo v luči zanesljivosti in odpornosti do napak.

Predstavljene vloge v opisanem modelu opravljajo specifične naloge. Glede na osnovni namen tega dela obstajajo vloge za pridobivanje podatkov, procesiranje pretočnih podatkov, napredno filtriranje, usmerjanje k naročnikom, analiziranje in vizualni prikaz rezultatov analiz.

1.1 Pregled področja

Hou, Huang, et. al. v [7] podajo teoretične osnove za vzpostavitev detekcije srčnega utripa (heartbeat) med več računskimi enotami in njihove implikacije za porazdeljeno okolje, ki se danes uporablja v primeru oblaka. Čeprav se njihovo delo ne dotika problemov, ki izhajajo iz visoko latentnega okolja

oblaka, v svojem delu dobro raziščejo in rešujejo potencialne težave z detekcijo utripa v primerih večjega števila procesnih enot, ki opravljajo generalizirane operacije. Hkrati predstavijo model, ki omogoča detekcijo srčnega utripa v primeru uporabe ene nadrejene in več podrejenih instanc samo z uporabo stanj in argumentirajo povečanje skalabilnosti v primerjavi z namenskimi dvojnimi (nadrejena, podrejena) vlogami instanc.

Warneke in Kao se v [8] ukvarjata s paralelizacijo podatkovnega procesiranja v oblaku in primerjata sistem Nephele z zmožnostmi sistema Hadoop, predvsem v smislu učinkovitosti paralelnega procesiranja na problemih, ki so MapReduce kompatibilni. V njunem delu predstavita ogrodje, ki omogoča optimalnejšo izbiro virov v oblaku glede na tip naloge, ki jo je potrebno rešiti in zaključujeta z možnostmi samodejnega povečevanja računskih virov. Paralelizacijo dosegata s povečevanjem števila virtualnih okolij, ki se samodejno vzpostavijo glede na zahteve dotičnega MapReduce procesnega vzorca. Tudi Shiraz, Gani, et. al. v [9] podajajo pregled obstoječih ogrodij za porazdeljeno aplikacijsko procesiranje v povezavi s pametnimi napravami in izvajanjem v oblaku. Dotikajo se predvsem problema paralelizacije, ki nastaja zaradi povečanega povpraševanja po procesiranju zaradi rasti števila mobilnih (in IoT) naprav, ki vsakodnevno dostopajo do interneta [10].

Shah in Kulkarni v [5] implementirata Storm, ki je v današnjem času gonilna tehnologija za omrežjem Twitter in predstavlja de-facto performančni standard na področju specifične implementacije tekstovnega usmerjanja sporočil od založnikov k naročnikom. Njuno delo predstavlja osnovo za usmerjanje sporočil na globalni skali z možnostjo porazdeljevanja bremena med več usmerjevalci. Pričujoče magistrsko delo se v določeni meri naslanja na koncept implementacije Storm usmerjevalnika, predvsem v področju usmerjanja sporočil oziroma rezultatov iskalnih kriterijev, vendar ga generalizira s poljubnimi (tipiziranimi) vrstami sporočil.

Xu, Cui, Wang et. al. podajajo pristope k kontroliranju kvalitete storitev [6] v primeru večjega števila izvajanj kompleksnih delovnih tokov v oblaku in ponudijo osnovo za razporejanje delovnih nalog glede na prioritete njihovih odjemalcev. S pomočjo primerov in eksperimentov prikažejo, da je mogoče s pametno strategijo razdeljevanja povečati prepustnost delovanja sistema.

Posebno pozornost podajajo zagotavljanju optimalnega razdeljevalnega algoritma, ki zagotavlja večjo prepustnost v povezavi s prioritetno shemo odjemalcev. To omogoča več možnosti v fazi razdeljevanja saj odjemalci z nižjo prioriteto (v lestvici kvalitete storitve) ne pričakujejo rezultatov enako frekventno, kot tisti z višjo prioriteto.

Banino, Beaumont, Carter et. al. se v [4] ukvarjajo z alokacijo velikega števila enako velikih nalog na heterogene procesne enote, ki so sposobne različnih hitrosti procesiranja. Njihov optimizacijski algoritem prilagaja razporejanje glede na zasedenost procesnih enot in njihovo ocenjeno hitrost procesiranja tako, da upošteva količino časa, ki je posledica komunikacije z drugimi procesnimi enotami in količino časa, ko procesor opravlja delo. Algoritem z linearnim programiranjem v polinomskem času najde optimalno razporeditev nalog, hkrati pa avtorji podajo teoretično primerjavo računske moči v primeru uporabe drevesno usmerjene platforme z močjo poljubno strukturiranih platform. Uporabnost njihovega teoretičnega modela je omejena le v smislu enakosti velikosti računskih nalog, kar je v realnih problemih težko zagotoviti.

Pregled tehnologij, gruč, virtualizacijskih mehanizmov in izračunavanja v oblaku podajajo Hwang, Dongarra in Fox v knjigi Distributed and Cloud Computing – From Parallel Processing to the Internet of Things [11], ki podaja osnovne mehanizme, algoritme in pregled pristopov na področju porazdeljenega izračunavanja. Delo predstavlja tudi naprednejše koncepte porazdeljevanja bremena in podaja celovit in moderen pregled, ki pokriva združevanje visoko performančnega izračunavanja, porazdeljevanja dela z mehanizmi v oblaku, virtualizacije in mrežnega izračunavanja (grid computing). Avtorji jasno prikazujejo poznavanje aplikacijskih in tehnoloških trendov, ki oblikujejo prihodnost računalništva na dotičnem področju.

1.2 Pomembnost porazdeljenega procesiranja

Zmožnost porazdeljenega procesiranja je za določeno problemsko domeno koristna iz več vzrokov. V tem primeru se osredotočamo na problemsko domeno, ki jo rešujemo v tem magistrskem delu in je opisana v poglavju 3.

V našem primeru je zmožnost porazdeljenega procesiranja pomembna iz treh vzrokov:

(1) Možnost skalabilnosti

Procesiranje, ki se izvaja v oblaku ima visoke zahteve po elastičnosti, tj. možnosti hitrega prilagajanja uporabljenih procesnih virov v času, ko je obremenitev (load) velika [3]. Elastičnost v tem smislu pomeni zmožnost hitrega povečevanja in zmanjševanja virov glede na količino zahtev po procesiranju. Možnost skalabilnosti ima zato pomembno vlogo, saj omogoča povečevanje in zmanjševanje števila procesnih enot v relativno kratkem času. Pravilna izvedba povečevanja skalabilnosti znotraj modela porazdeljenega procesiranja prinaša dodatne omejitve, ki so opisane v naslednjih poglavjih.

(2) Možnost geografskega porazdeljevanja

Definicija: Dostopnik (accessor)

Dostopnik je odjemalski proces, ki dostopa do določenega internetnega vira v smislu modela procesiranja opisanega v tem magistrskem delu. Internetni vir je dosegljiv na nekem internetnem naslovu in ni vezan na specifičen transportni protokol (HTTP, HTTPs, TCP/IP, UDP, itd.). Internetni vir lahko od dostopnika zahteva dodatne avtentikacijske podatke in lahko vedno ugotovi njegov internetni naslov IP.

Dostopniki prihajajo do internetnih informacijskih virov iz različnih geografskih, in posledično, različnih izvornih naslovov IP. Geografsko porazdeljevanje je pomembno ravno zaradi zagotavljanja možnosti dostopa iz različnih naslovov IP, saj je tako mogoče doseči bistveno večjo prepustnost iskanja na internetnem viru.

Velika večina pomembnejših spletnih strežnikov (Microsoft IIS, Apache, nginx, GWS, itd.) in ponudnikov vsebin na internetu v današnjem času omejuje prepustnost (throttling) na nivoju avtenticiranega uporabnika in njegovega izvornega naslova IP. Enako večkratne dostope obravnavajo tudi ponudniki popularnih družbenih omrežij [12] [13]. Z geografskim porazdeljevanjem v oblaku se je mogoče temu izogniti.

(3) Odpornost na napake

V primeru, ko se definicije operacij (opredelitev dela) nahajajo na lokaciji, ki je dosegljiva vsem procesnim enotam, izvajanje enega tipa

operacije na več računskih enotah pozitivno vpliva na odpornost na napake. V primeru odpovedi ene ali več procesnih enot delo prevzamejo delujoče, celoten sistem pa v vmesnem času deluje z zmanjšano zmogljivostjo.

1.3 Pomen skalabilnosti

V splošnem se višja skalabilnost dosega z uporabo a) povečevanja moči enotnega vira procesiranja (scale-up) ali b) povečevanjem števila virov procesiranja (scale-out).

Definicija: Moč vira procesiranja

Moč vira procesiranja predstavlja splošno zmožnost procesiranja nekega računskega vira. Vanjo štejemo tako število fizičnih procesorjev, njihovo hitrost, število niti v procesorjih, diskovni prostor in njegovo hitrost ter količino pomnilnika.

Prvi pristop, torej povečevanje moči enotnega vira procesiranja omogoča hitrejše izvajanje ne-paralelizabilnih in paralelizabilnih operacij. Operacije, ki zaradi svoje narave ne omogočajo paralelizacije ali zanje paralelizacija predstavlja velik tehnični izziv, z večjo močjo vira pridobijo na hitrosti izvajanja. Paralelizabilne operacije pridobijo iz istega vzroka, hkrati pa skoraj vsi računski viri v oblaku v današnjem času omogočajo paralelizacijo niti, kar implicitno omogoča tudi povečevanje hitrosti paralelizabilnih operacij. Težava pristopa s povečevanjem moči enotnega vira procesiranja je v njegovi geografski in naslovni (IP) odvisnosti, saj v našem modelu ne omogoča maksimalnega izkoristka moči, ki je na voljo.

Po drugi strani je pristop s povečevanjem števila virov procesiranja primernejši za naš model, saj omogoča geografsko porazdelitev, hkrati pa je mogoče takim računskim virom vedno tudi povečati moč.

Definicija: Ekonomsko smiseln način

Ekonomsko smiseln način v kontekstu tega dela pomeni cenejši, hitrejši za razvoj, primernejši in učinkovitejši za vzdrževanje rešitve, bolj odporen na odpovedi in z manjšimi tveganji za kršitev pogodb SLA (service level agreement).

Eno od vprašanj, ki si ga zastavljamo v magistrski nalogi je torej: Ali je mogoče z več manjšimi računskimi viri, na ekonomsko smiseln način v današnjem času ponudnikov procesiranja v oblaku, in s povečevanjem števila virov procesiranja izvesti več operacij, kot v primeru povečevanja moči enotnega vira procesiranja?

1.4 Ekonomski vidiki

Ponudnikov storitev procesiranja v oblaku je zelo veliko. V tej nalogi se v implementacijskem delu fokusiramo na specifičnega ponudnika, ki ima po našem mnenju trenutno najboljšo kombinacijo geografske porazdelitve in tehnološke prednosti na področju razvoja kompleksnih rešitev v oblaku – podjetje Microsoft Corporation[1]. Ponudba oblačnih storitev tega podjetja vključuje orodja, ki pokrivajo celotni življenjski cikel razvoja rešitve: od konceptualizacije in načrtovanja, preko razvoja, testiranja, obvladovanja razvojne skupine in izvorne kode, do gostovanja v oblaku in možnosti skaliranja.

S tehničnega gledišča je porazdeljeno procesiranje na več koncih sveta mogoče izvesti z večino velikih ponudnikov oblačnih storitev (Microsoft, Amazon, Google, itd.). Razlike nastajajo v podrobnostih, ko je potrebno celostno rešitev pripeljati na trg v relativno kratkem roku ter z omejenimi človeškimi viri ter tako storitev vzdrževati na točki pozitivnega poslovnega modela. Zaradi tehnične različnosti ponudbe med vodilnimi igralci na trgu oblačnih storitev, je neposredna primerjava izvedbe kompleksnih rešitev zelo otežena. V pričujočem delu se zato ne ukvarjamo z argumentacijo kateri ponudnik bi bil bolj primeren za posamezni sklop modela, ampak se znotraj omejitev določenega ponudnika odločamo za najboljši možen pristop k implementaciji teoretičnega modela predstavljenega v poglavju 2 .

1.5 Trenutno stanje industrije

Sprejemanje tehnologij oblaka se je v zadnjih letih pospešilo. Zaradi povečane ponudbe in večje agilnosti storitev, vedno več podjetij premika svoje

[1] Microsoft Azure, http://azure.microsoft.com

aplikacije iz lastnih podatkovnih centrov v oblak. Rast oblaka pospešujejo tudi nove stranke v segmentu malih in srednjih podjetij (SME, small and medium enteprises), ki se za oblak odločajo predvsem zaradi manjše začetne investicije in visokih pričakovanj za prihodnje storitve v oblaku [14].

Pospešek prehoda v oblak je bil v letu 2015 posebno očiten [15], saj je bil samo trg infrastrukture kot storitve (IaaS, Infrastructure as a Service) vreden preko 16 milijard ameriških dolarjev.

Na področju IaaS gre za trg, kjer je doslej v veliki meri prevladoval Amazon s storitvijo AWS. Vzrok je predvsem zgodnji vstop na trg[2], vendar kljub prevladi je Microsoft širil ponudbo in vzpostavil ogromno globalno mrežo storitev v oblaku. Čeprav v času pisanja naloge sama količina računske moči podjetja Microsoft še ni tako velika kot AWS, je vseeno približno dvakrat večja od naslednjega najbližjega tekmeca. V ponudbi storitev sta tekmeca trenutno izenačena, v obvladovanju celotnega življenjskega cikla razvoja rešitve, pa ima Microsoft Azure prednost predvsem zaradi homogene in poenotene izkušnje, ki ni zanemarljiva.

Izbira enega ponudnika storitev v oblaku je odvisna od zahtev posamezne stranke in vrste obremenitev, ki jih izvaja. V veliko primerih se stranke odločajo za hibridne rešitve, kjer določeni deli rešitve tečejo v oblaku enega ponudnika, druge pa pri drugem ponudniku.

S stališča ponujenih storitev v oblaku se obe podjetji precej pokrivata, saj ponujata podobne zmožnosti predvsem v smeri izračunavanja (compute), hrambe (storage) in omrežja (networking). Obe podjetji omogočata hitro vzpostavitev storitve, samodejno skaliranje, varnost, visoko spoštovanje sporazuma o ravni storitev (service level agreement) in upravljavska orodja, ki omogočajo preprostejše obvladovanje kompleksnih rešitev.

Na področju MapReduce algoritmov tako Amazon kot Microsoft ponujata specifične implementacije tehnologije, prvi v obliki AWS Elastic Map Reduce in drugi kot storitev HDInsight. Obe podjetji sta v svoj nabor storitev v oblaku dodali storitve za strojno učenje, storitve interneta stvari (IoT), podporo za mobilne aplikacije in visoko performančne virtualne računalnike

[2] Amazon AWS je na voljo od leta 2006, Microsoft Azure od leta 2010.

(virtual macines), ki omogočajo izvedbo hitrega izračunavanja za stranke, ki ga potrebujejo.

Ob koncu leta 2015 so vsi največji ponudniki storitev v oblaku ponudili tudi podporo za tehnologijo zabojnikov Docker (Docker containers), ki omogoča dodatni nivo abstrakcije pri izvajanju programja. V tem pogledu je Microsoft v svojem strežniškem operacijskem sistemu Windows Server 2016 omogočil celo gostovanje slik Docker (Docker images) znotraj lokalnih podatkovnih centrov (on-premise data center), kar omogoča dodaten korak k zanesljivemu in zaupanja vrednemu hibridnemu oblaku.

1.6 Motivacija in cilji

Osnovni cilj magistrskega dela je torej definicija modela za pridobivanje podatkov iz poljubnih naslovljivih izvorov, procesiranje ter analitika, ki je aplikativna v več uporabniških rešitvah. Želimo implementacijo modela z visoko stopnjo skalabilnosti, primerno tudi za povpraševalne tokove, ki omogočajo povpraševanje po specifičnih, vnaprej določenih virih.

Raziskovani model predstavlja osnovo za masovno procesiranje podatkov v oblaku s pristopi, ki so uporabni pri večini ponudnikov in niso specifično vezani na tehnologijo ali ponudnika.

V magistrsem delu predstavljamo načine za porazdeljevanje dela med sebi enakimi procesorskimi entitetami (vlogami), ki jih upravlja vodilna (ali nadrejena) instanca. Ukvarjamo se z učinkovitimi načini za izbiro vodij, ki svoje delo izvajajo na različnih porazdeljenih lokacijah ter načini za zagotavljanje vsaj enega delujočega vodje. Hkrati opredeljujemo načine za učinkovito zagotavljanje idempotentnosti izvajanja operacij, kar ima v visoko latentnem okolju izvajanja procesiranja v oblaku pomembno vlogo v luči zanesljivosti in odpornosti do napak.

1.7 Struktura dela

Takoj po uvodu, v poglavju 2 (Konceptualni model) opisujemo uporabljene koncepte skalabilnega procesiranja. Poglavje se osredotoča na konceptualni

model, njegovo sestavo, faze in vloge, ki jih opravljajo posamezne procesne enote.

V poglavju 3 (Problemska domena), podajamo opis problemske domene na kateri smo testirali predmet oziroma témo magistrske naloge. Problemska domena je generalizirana ter z veliko možnostmi razširjanja in potencialnih dodatnih analiz.

V poglavju 4 (Model pridobivanja podatkov) podrobneje opišemo model pridobivanja podatkov. Glede na potrebo po skalabilnem procesiranju potrebujemo pridobiti masovno količino podatkov v kratkem času, kar pomeni, da je potrebno doseči visoko stopnjo skaliranja tudi v točki pridobivanja podatkov.

Poglavje 5 (Model izločanja, usmerjanja in filtriranja podatkov) opredeljuje izvedeni model filtriranja pridobljenih podatkov s koncepti, ki so primerni za prikazano problemsko domeno.

V poglavju 6 (Model porazdeljevanja bremena) podrobneje opišemo splošne koncepte modela za porazdeljevanje bremena, ki vključujejo uporabljene in preizkušene pristope.

Poglavje 7 (Model vizualizacije pridobljenih analiz) prikazuje možnosti in koncepte vizualizacije rezultatov pridobljenih analiz nad izvornimi podatki.

Zadnje poglavje, poglavje 8 (Sklepne ugotovitve) podaja sklepe in možnosti izboljšav, ki smo jih identificirali tekom študija in izvedbe magistrskega dela.

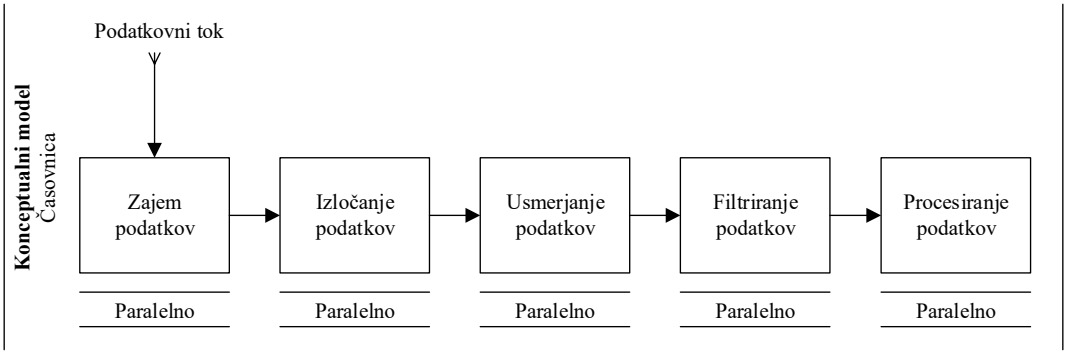
2 Konceptualni model

V poglavju predstavljamo konceptualni model za pridobivanje, usmerjanje, filtriranje in procesiranje masovne količine podatkov iz konstantnih in povpraševalnih podatkovnih tokov. Model podatke pridobi, izloči neprimerne, usmerja uporabne rezultate, jih filtrira in končno procesira zadetke z analizami, ki so uporabne glede na uporabljeno problemsko domeno.

Konceptualni model je skalabilen v vseh fazah izvajanja. Skalabilnost dosega z različnimi mehanizmi, od fizičnega porazdeljevanja na več računskih instanc, do paralelnega izvajanja znotraj posamezne instance. Model predvideva uporabo koncepta nadrejenih-podrejenih instanc in voljenje v primerih, ko vodilna instanca ni na voljo ali ne reagira v določenem času.

Konceptualni model ima več faz, ki predstavljajo obravnavanje podatkov ob različnih časih – od nastanka, do shranjevanja surovih podatkov in podatkov analiz, ki iz surovih podatkov nastanejo.

Časovnica obravnavanja pridobljenih podatkov je naslednja:



Slika 2.1: Časovnica obravnavanja podatkov

Konceptualno se celoten vhodni podatkovni tok zajame, podatki se izločijo, usmerijo, filtrirajo in na koncu procesirajo skladno z analizami definiranimi

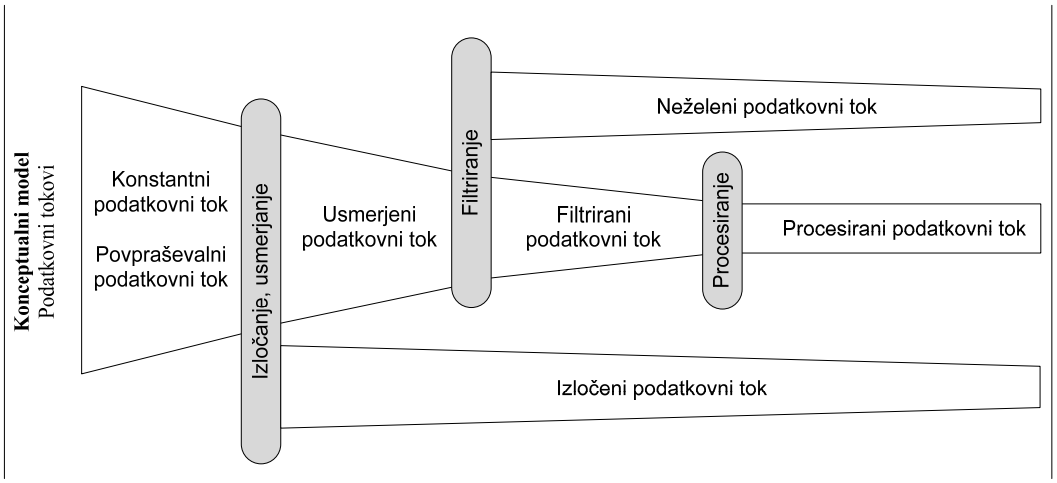
v modelu za porazdeljevanje bremena analiz. Iz zgornje slike je razvidno, da je celotno procesiranje v modelu sestavlja *procesni cevovod*, kjer se podatki iz ene faze prenesejo v naslednjo fazo procesiranja.

V sliki 2.1 vizualizacija podatkov in izračunanih analiz ni podana, ker je čas prikaza (vizualizacije) analiz nepovezan s časom zajema in obvladovanja podatkov. Analize se prikazujejo kadarkoli po dokončanju faze procesiranja podatkov.

Konceptualni model modelira različne podatkovne tokove, ki so namenjeni opisu obvladovanja podatkov. Podatkovni tokovi so opisani konceptualno in obstajajo samo v času procesiranja znotraj procesnega cevovoda.

2.1 Podatkovni tokovi

Celotni *vhodni podatkovni tok* se deli na *konstantni podatkovni tok* (več v poglavju 2.2.1) in *povpraševalni podatkovni tok* (poglavje 2.2.2). Vhodni tok se v fazi usmerjanja razdeli na *usmerjeni podatkovni tok* (podatke, ki preživijo izločanje in se usmerijo naprej) in *izločeni podatkovni tok* – podatke, ki se izločijo iz nadaljnjega procesiranja. Usmerjeni podatkovni tok se v fazi filtriranja razdeli na *filtrirani podatkovni tok* in *neželeni podatkovni tok*, glede na nastavljene filtre iskalnih kriterijev. Vsi podatki, ki preživijo filtriranje so primerni za procesiranje in po fazi procesiranja sestavljajo *procesirani podatkovni tok*.



Slika 2.2: Podatkovni tokovi

Definicija: Uporabni podatki

Uporabne podatke sestavljata filtrirani in procesirani podatkovni tok. Podatki konstantnega, povpraševalnega in usmerjenega podatkovnega toka so pogojno uporabni, medtem ko so podatki izločenega in neželenega podatkovnega toka neuporabni.

V naslednjih poglavjih opisujemo modele pridobivanja, izločanja usmerjanja in filtriranja podatkov ter modele porazdeljevanja bremena in vizualizacije. Vsi modeli delujejo nad podatkovnimi tokovi orisanimi na sliki 2.2.

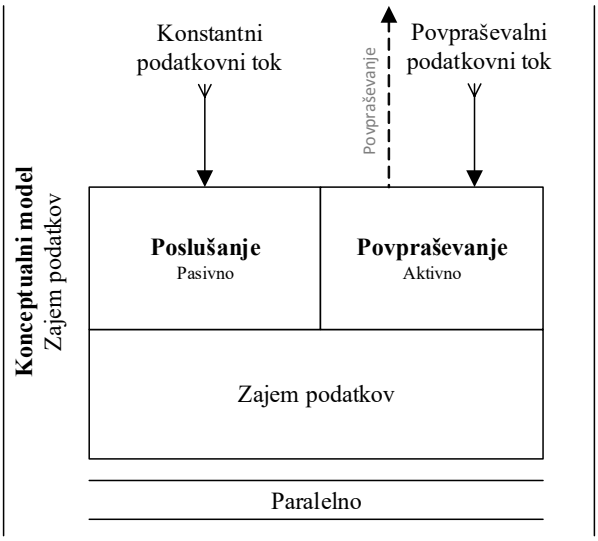
2.2 Model pridobivanja podatkov

V tem poglavju podajamo konceptualni model pridobivanja podatkov, ki je namenjen *pasivnemu sprejemu* in *aktivnemu povpraševanju* po rezultatih. Razlika med njima je v načinu izmenjave podatkov, hitrosti sprejema in možnostih, ki jih posamezni način ponuja.

S stališča izmenjave podatkov gre za dva mehanizma:

(1) Potisni mehanizem (push mechanism)
Ponudniki podatkov potiskajo podatke k odjemalcem. Odjemalci podatke pasivno sprejemajo.

(2) Vlečni mehanizem (pull mechanism)
Odjemalci aktivno povprašujejo po novih podatkih. Ponudniki jih pošljejo po prejemu povpraševanja.



Slika 2.3: Konstantni in povpraševalni podatkovni tok

Skupni podatkovni tok, ki nastaja na podlagi obeh mehanizmov in prihaja v sistem procesiranja, je segmentiran v dva definirana tipa tokov: *konstantni podatkovni tok* in *povpraševalni podatkovni tok*.

Naj bo *data* enota podatka zajema, torej najmanjša enota, ki jo zajamemo, filtriramo, usmerjamo ali procesiramo.

$$data = \begin{array}{|c} property_1 \\ property_2 \\ \cdots \\ property_n \end{array} \tag{2.1}$$

V tem primeru je $property_k$ lastnost podatka, ki je definirana kot dvojica $(name, value)$, kjer je *name* ime lastnosti in *value* njena vrednost[3].

Primer tipičnega zajetega podatka (rezultata) *data* z nekaj dodatnimi lastnostmi je podan spodaj:

$$data = \left| \begin{array}{c} source\text{: } 'twitter' \\ user\text{: } 'matevzg' \\ mentions\text{: } '' \\ reply\text{: } 0 \\ id\text{: } 270464853170251672 \\ followers\text{: } 710 \\ following\text{: } 231 \\ text\text{: } 'to\ je\ preprost\ primer' \end{array} \right.$$

Slika 2.4: Primer zajetega rezultata

Lastnost konstantnega toka podatkov je, da je predstavljen v obliki toka (stream) in je tesno časovno sinhroniziran z izvorom. Latenca (razlika med časom nastanka podatka in časom zajema) L_Kv konstantnem toku je enaka vsoti transportnega časa od izvora do točke zajema in porabljenega časa na aktivni opremi[4]. Opišemo jo kot:

$$L_K = t_T + t_O \tag{2.2}$$

Kjer je t_T transportni čas, t_O pa čas aktivne opreme, ki je uporabljena pri transportu.

[3] Klasična dvojica ime-vrednost (name-value pair).

[4] Med aktivno opremo v tem kontekstu prištevamo usmerjevalnike, stikala, strežnike, požarne zidove in drugo opremo IT, ki sodeluje pri transportu podatkov.

Povpraševalni tok podatkov, torej drugi tip podatkovnega toka, je izveden z mehanizmom povpraševanja (pull mechanism) k izvornemu ponudniku podatkov. Latenco L_P, za primer povpraševalnega toka, zapišemo kot:

$$L_P = t_T + t_O + t_P \tag{2.3}$$

Kjer je t_P čas (ali dolžina) povpraševalne periode. Očitno je, da je $L_P \gg L_K$ v primeru, ko je čas nastanka novih podatkov grobo neusklajen s periodo povpraševanja. Enačba (2.3) sicer predstavlja maksimalni čas $L_{P_{max}}$, kjer je neusklajenost med časom nastanka podatkov in časom povpraševanja maksimalna. V nasprotnem primeru dobimo $L_{P_{min}}$, ki je enak L_K. Razmerje med L_P in L_K lahko definiramo kot:

$$L_K \leq L_{P_{min}} \ll L_{P_{max}} \tag{2.4}$$

Enačba (2.4) pove, da je latenca konstantnega toka vedno manjša ali enaka latenci povpraševalnega toka.

Točke zajema obeh tipov podatkovnih tokov so implementirane z *dostopniki* (glej poglavje 1.2). Ti predstavljajo kos programja, ki implementira poslušanje v primeru konstantnega toka podatkov in aktivno povpraševanje v primeru povpraševalnega toka podatkov.

2.2.1 Konstantni tok podatkov

Tok podatkov (tako konstantni - T_k, kot povpraševalni - P_k) je definiran v obliki strukture, ki je sestavljena iz urejene m-terice $DSET_{T_k}$:

$$DSET_{T_k} = (data_1, data_2, \dots, data_m) \tag{2.5}$$

Kjer je m kardinalnost podatkovne m-terice (število zajetih podatkovnih struktur), T_k k-to časovno obdobje zajema, definirano na časovnem intervalu $[t_a, t_b)$ in je t_a začetni čas, t_b pa končni čas zajema toka. Za vsak T_k vedno velja:

$$t_a < t_b \tag{2.6}$$

Kardinalnost m-terice $DSET_{T_k}$ je spremenljiva, saj je odvisna od števila podatkovnih struktur, ki so na voljo v konstantnem toku ob časovnem obdobju T_k.

Časovna obdobja T sestavljajo urejeno n-terico, kjer velja:

$$T = (T_1, \dots, T_k, \dots, T_n) \tag{2.7}$$

Skladno z (2.6) model zajema predlaga, da je trajanje obdobja $t_k = t_b \quad t_a \geq$ 1000 ms (1 sekunda), kar je neposredna posledica želje po pogostem dotoku novih rezultatov v primeru, če so ti na voljo v konstantnem toku podatkov. Razlika med t_b in t_a torej določa tudi periodo prehoda v naslednjo aktivnost modela (filtriranje) in omejuje spodnjo mejo frekvence osveževanja rezultatov znotraj sistema.

Bolj frekventno kvantificiranje časovnih obdobij (na manj kot 1000 ms) je s stališča uporabniške izkušnje nepomembno, hkrati pa nepotrebno obremenjuje procesne enote, zato predlagamo vrednosti skladne s omenjenimi.

V primeru, ko se v časovnih obdobjih T_k, kjer je $1 \leq k \leq n$ iz konstantnega toka zajame povprečno $\overline{m}$ podatkov (povprečna kardinalnost m-terice $DSET_{T_k}$ je enaka $\overline{m}$ ali $\#DSET_{T_k} = \overline{m}$), velja:

$$N_T = \overline{m} \times n = \#DSET \tag{2.8}$$

Kjer je N_T število rezultatov pridobljenih v časovnih obdobjih od 1 do n.

Analogno naj bo m_k število pridobljenih rezultatov iz konstantnega toka znotraj časovnega obdobja T_k in t_k njegovo trajanje. Čas procesiranja vseh podatkov $data_k$, kjer je $1 \leq k \leq m$, mora biti manjši od t_k. Povprečni čas procesiranja posameznega podatka $data_k$ tako ne sme preseči $\overline{t}$:

$$\overline{t} \leq \frac{t_k}{m_k} \tag{2.9}$$

Sledi, da za vsako časovno obdobje velja, da mora(jo) procesor(ji) uspeti s procesiranjem vseh pridobljenih podatkov v T_k najmanj v času t_{max}:

$$\forall\ T_k\text{: } t_{max} \leq (\tfrac{t_k}{m_k} \times m), \text{ kjer je } m = \#DSET_{T_k} \tag{2.10}$$

Obvladljivost toka je zato odvisna od časa procesiranja podatkov v časovnih obdobjih.

Definicija: Obvladljivost konstantnega podatkovnega toka

Konstantni podatkovni tok je ***obvladljiv*** *v primeru, ko je povprečni čas procesiranja posameznega časovnega obdobja T_k manjši ali enak t_{max} in neobvladljiv obratno.*

Tipično obdobje T_k traja *nekaj sekund* – kar pa ni zahtevano, saj model dovoljuje tudi daljša časovna obdobja, v odvisnosti od zmožnosti procesiranja in pomnilnika, ki je na voljo.

2.2.2 Povpraševalni podatkovni tok

Skladno z (2.5) je tudi povpraševalni tok P_k definiran kot urejena m-terica pridobljenih podatkov $DSET_{P_k}$:

$$DSET_{P_k} = (data_1, data_2, \dots, data_m) \tag{2.11}$$

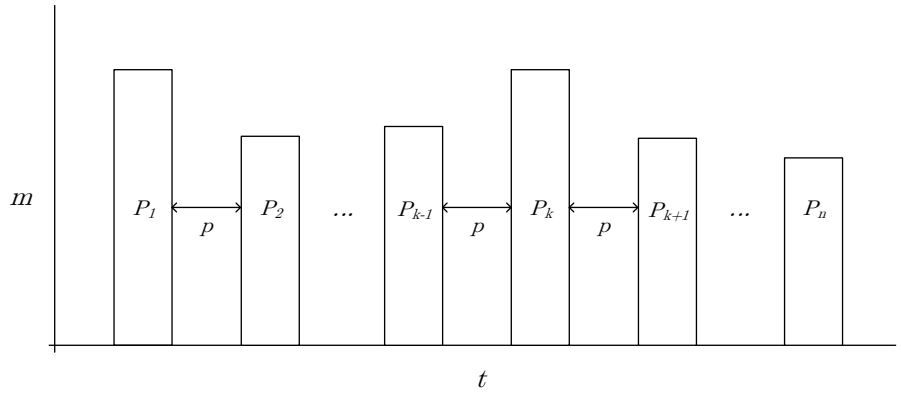
Povpraševanja P sestavljajo urejeno n-terico:

$$P = (P_1, \dots, P_k, \dots, P_n) \tag{2.12}$$

Kjer je p perioda povpraševanja, P_k povpraševanje, t_{P_k} pa čas povpraševanja za katerega velja:

$$t_{P_{k\ 1}} < t_{P_k} < t_{P_{k+1}} \text{ in } t_{P_k} = (t_{P_{k\ 1}} + p) \text{ in } t_{P_{k+1}} = (t_{P_k} + p) \tag{2.13}$$

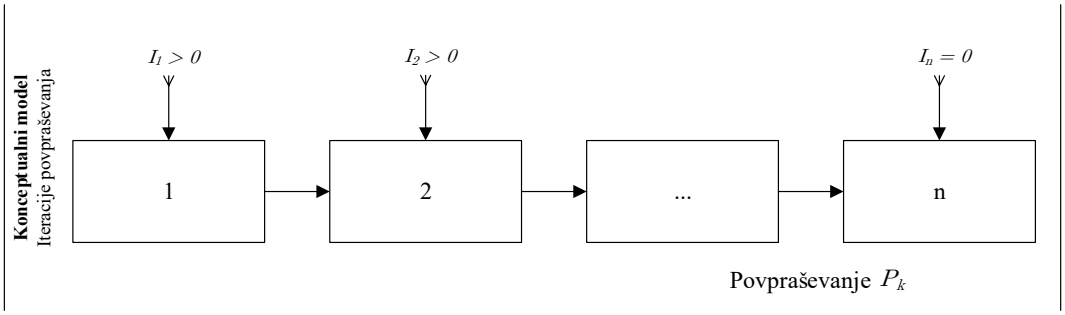
Povpraševanja P torej definirajo časovno ekvidistančno vrsto, kjer se naslednje povpraševanje zgodi vsako periodo p.



Slika 2.5: Ekvidistančna vrsta povpraševanj

Povpraševalni tok podatkov je bolj obvladljiv od konstantnega, ker je maksimalna kardinalnost podatkovne m-terice nastavljiva. Pri odgovoru na povpraševanje P_k se model za pridobivanje podatkov lahko odloči za omejevanje količine zajetih podatkov in jo omeji z vrednostjo $N_{P_{max}}$. Povpraševanje po rezultatih nato nadaljuje v P_{k+1}.

Vsako povpraševanje P_k je lahko sestavljeno iz več iteracij, ki se izvajajo zaporedno. Naj bo I_i število pridobljenih rezultatov v iteraciji i povpraševanja P_k.



Slika 2.6: Povpraševanje P_k

Iteracije znotraj enega povpraševanja se ponavljajo dokler je vrnjeno število rezultatov v iteraciji večje od nič. Tako velja:

$$\forall\ j \in \{0..i\}\colon I_j > 0 \tag{2.14}$$

Povpraševanje P_k se prekine takoj, ko iteracija i ne vrne rezultatov (vrne nič rezultatov). Takrat velja:

$$\forall\ j \in \{0..i \quad 1\}\colon I_j > 0 \text{ in } I_i = 0 \tag{2.15}$$

Naj bo $N_{P_{max}}$ največje število rezultatov, ki jih lahko pridobi povpraševanje P_k. Naj bo N_{P_k} število rezultatov, ki jih pridobi povpraševanje P_k. Naj bo I_{max} največje število rezultatov, ki jih povpraševanje pridobi v eni iteraciji.

Potem velja:

$$N_{P_k} = \min(\sum_{i=0}^{n\ \ 1} min(I_i, I_{max})\,, N_{P_{max}}) \tag{2.16}$$

Največji dovoljen čas procesiranja t_{max} povpraševanja P_k je omejen z dolžino periode p in je definiran kot:

$$\forall\ P_k\colon t_{max} \leq p \tag{2.17}$$

Enačba (2.17) zahteva, da mora biti čas procesiranja posameznega povpraševanja P_k krajši od časa trajanja periode.

V primeru povpraševalnega toka podatkov torej čas t_{max} določa obvladljivost toka.

Definicija: Obvladljivost povpraševalnega podatkovnega toka

Povpraševalni podatkovni tok je ***obvladljiv*** *v primeru, ko je povprečni čas procesiranja posameznega povpraševanja P_k manjši ali enak t_{max} oziroma manjši ali enak od periode p in neobvladljiv obratno.*

Večja kot je perioda p za posamezni iskalni kriterij, večja je zakasnitev novih rezultatov. Če privzamemo, da imamo v modelu i iskalnih kriterijev, torej $S = \{s_1, ..., s_i\}$, potem lahko *iskalno prepustnost* pri večjem številu iskalnih kriterijev, z uporabo več instanc, definiramo kot:

Definicija: Iskalna prepustnost

Iskalna prepustnost je število iskalnih kriterijev, za katere istočasno iščemo nove rezultate.

Višja iskalna prepustnost pozitivno vpliva na maksimalni interval zakasnitve iskalnega kriterija, ki je definiran kot razlika med zadnjim časom iskanja in trenutnim časom.

Naj bo t_{s_i} zadnji čas iskanja za iskalni kriterij s_i in t_{now} trenutni čas. Interval zakasnitve d_{s_i} iskalnega kriterija s_i je torej:

$$d_{s_i} = t_{now} \quad t_{s_i} \tag{2.18}$$

Interval zakasnitve za vse iskalne kriterije, torej za celoten sistem, tako definiramo kot d:

$$d = \max_{j=1}^{i}(d_{s_i}) \tag{2.19}$$

2.3 Model izločanja, usmerjanja in filtriranja podatkov

V tem poglavju definiramo konceptualni model za izločanje, usmerjanje in filtriranje podatkov. V podpoglavjih 2.3.1, 2.3.2 in 2.3.3 podrobneje podajamo faze izločanja, usmerjanja in filtriranja.

Faze znotraj modela si sledijo v naslednjem vrstnem redu:

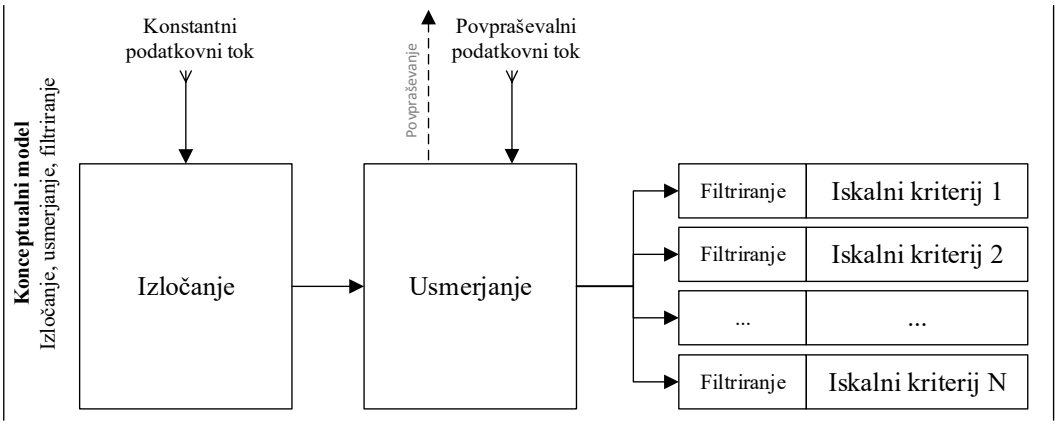
(1) Faza izločanja
(2) Faza usmerjanja
(3) Faza filtriranja

Ad (1) izloči večino podatkov saj deluje nad konstantnim podatkovnim tokom, ki je bistveno širši od želenih podatkov.

Ad (2) usmeri pridobljene podatke na ustrezne iskalne kriterije. Usmerjanje je izvedeno z mehanizmom pub-sub za konstantni podatkovni tok in trivialno za povpraševalni podatkovni tok, kjer so iskalni kriteriji znani vnaprej.

Ad (3) filtrira podatke z dodatnimi filtri, ki so nastavljeni na vsakem iskalnem kriteriju.

Najprej se izvede izločanje (glej poglavje 2.3.1). Nato se izvede usmerjanje (poglavje 2.3.2) in na koncu filtriranje (poglavje 2.3.3).



Slika 2.7: Izločanje, usmerjanje, filtriranje

Izločanje podatkov[5] se lahko zgodi le v fazah izločanja in filtriranja. Faza usmerjanja ne izloča podatkov, ampak jih le pošilja v ustrezni iskalni kriterij.

S stališča *uniformnosti* sta izločanje in usmerjanje globalni operaciji in se izvajata nad vsemi podatki / rezultati, ki prihajajo v sistem. Filtriranje, po drugi strani, se izvaja nad specifičnimi rezultati kriterija, saj je definicija filtrov lahko različna za vsak kriterij.

Izločanje in filtriranje sta s stališča vpliva na sprejete podatke *invazivni*, saj podatke izpuščata, če so izpolnjeni pravi pogoji. Nasprotno je usmerjanje neinvazivna operacija.

Naslednja tabela definira lastnosti vseh treh faz glede uniformnosti in invazivnosti:

[5] Izločitev podatka pomeni, da se podatek / rezultat nadalje ne procesira in se zavrže.

faza	uniformnost	invazivnost
izločanje	vsi rezultati	izgubno
usmerjanje	vsi rezultati	neizgubno
filtriranje	samo rezultati kriterija	izgubno

Tabela 2.1: Uniformnost in invazivnost faz izločanja, usmerjanja in filtriranja

2.3.1 Faza izločanja

Faza izločanja je aplikativna le pri *konstantnem podatkovnem toku*, kjer se zavržejo vsi prejeti podatki, ki ne ustrezajo nobeni od definicij iskalnih kriterijev. S tega gledišča izločanje podatkov pri povpraševalnem toku ni mogoče, saj so povpraševanja po definiciji vedno namenjena obstoječemu iskalnemu kriteriju.

Naj bo množica S množica i iskalnih kriterijev, torej $S = \{s_1, \dots, s_i\}$. Naj bo W_{s_k} množica j ključnih besed (keywords), torej $W_{s_k} = \{w_1, \dots, w_j\}$ iskalnega kriterija s_k. Unija vseh ključnih besed W, na katere reagira faza izločanja je sestavljena iz ključnih besed določenih v vseh kriterijih, torej:

$$W = \bigcup_{k=1}^{i} W_{s_k} \tag{2.20}$$

Skladno s (2.5) je $DSET_{T_k} = (data_1, data_2, \dots, data_m)$ m-terica pridobljenih rezultatov v časovnem obdobju T_k. Naj bo $words_p = \{u_1, \dots, u_v\}$ množica besed v pridobljenem rezultatu $data_p$[6].

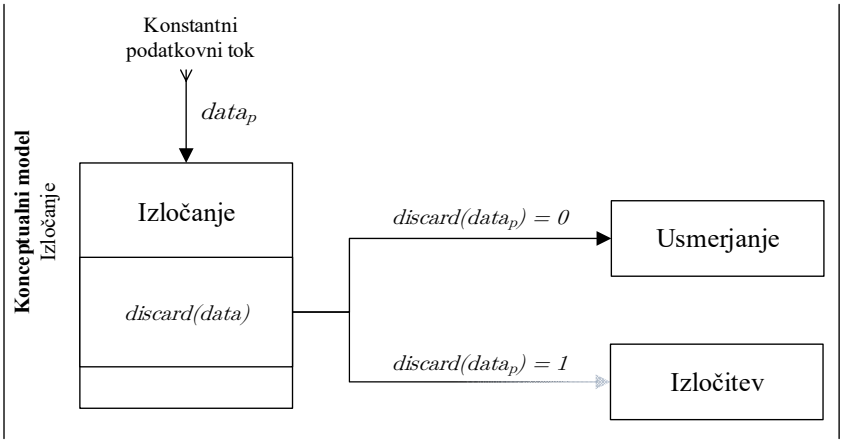
Potem je izločitvena funkcija ***discard*** naslednja:

$$discard(data_p) = \begin{cases} 0; \bigcap (words_p, W) \neq \\ 1; \bigcap (words_p, W) = \end{cases} \tag{2.21}$$

Izhod izločitvene funkcije je binaren. V primeru, ko obstaja presek med množico besed $words_p$ rezultata $data_p$ in množico ključnih besed W, se rezultat ne izloči; z drugimi besedami, rezultat $data_p$ se ohrani, če funkcija vrne vrednost 0 in izloči, če funkcija vrne 1.

Spodnja shema prikazuje fazo izločanja.

[6] Primer $words_p$ je: {'to, 'je', 'preprost', 'primer'}.



Slika 2.8: Faza izločanja za konstantni tok podatkov

Odstotek izločenih podatkov je odvisen od števila iskalnih kriterijev, ključnih besed, ki so v njih določene in splošne porazdelitve besed v konstantnem toku. Ker je verjetnost izločitve zelo odvisna od števila ključnih besed v kriterijih in téme preiskovanja, je napovedovanje verjetnosti izločitve zelo nezanesljivo.

Naj bo $P(I)$ verjetnost izločitve in $P(U)$ verjetnost usmerjanja. Potem velja:

$$P(I) = 1 \quad P(U) \tag{2.22}$$

Za večino *praktičnih primerov* velja tudi:

$$P(I) \gg P(U) \tag{2.23}$$

Enačba (2.23) pravi, da je verjetnost izločitve v primeru konstantnega toka v veliki večini primerov veliko večja od verjetnosti nadaljnjega usmerjanja. Možne so sicer situacije, ko to ne velja, vendar gre za izredne primere in zelo kratka časovna obdobja, ko je komunikacija na družbenih omrežjih omejena na globalno in specifično témo[7].

2.3.1.1 Možnost hrambe izločenih podatkov

Glede na veliko količino podatkov, ki lahko prihajajo v sistem[8], se hramba izločenih podatkov trenutno *ne predvideva*. V primerih, kjer je zahtevana

[7] Primeri so npr. gol v finalu svetovnega prvenstva v nogometu, potrditev izvolitve ameriškega predsednika.

[8] Konstantni tok podatkov družbenega omrežja Twitter (Twitter firehose) dnevno vsebuje povprečno 450 milijonov rezultatov, od katerih se jih velika večina izloči.

kasnejša analiza bi bila taka funkcionalnost sicer dobrodošla, vendar zaradi obremenitev, ki bi jih zahtevala, v modelu trenutni ni predvidena.

2.3.2 Faza usmerjanja

Faza usmerjanja je namenjena pošiljanju podatkov v prave iskalne kriterije in omogoča centralno posredovanje (brokering) vseh podatkov, ki prihajajo v sistem. Usmerjanje model obravnava ločeno za konstantni in povpraševalni tok predvsem zaradi preprostejše (in hitrejše) možnosti ugotavljanja pravilnega iskalnega kriterija v primeru povpraševalnega podatkovnega toka.

Usmerjanje podatkov je implementirano z mehanizmom založnik-naročnik (pub-sub) in je, z namenom paralelnega izvajanja, decentralizirano.

Način implementacije generičnega usmerjevalnika, ki je uporabljen tudi v tem primeru, podajamo v poglavju 5.2.

2.3.2.1 Usmerjanje konstantnega toka

Naj bo množica S množica i iskalnih kriterijev, torej $S = \{s_1, \ldots, s_i\}$. Množica vseh besed na katere reagira faza usmerjanja je W in je enaka definiciji v enačbi (2.20).

$$W = \bigcup_{k=1}^{i} W_{s_k} \tag{2.24}$$

Kjer je $W_{s_u} = \{w_1, \ldots, w_j\}$ množica ključnih besed iskalnega kriterija s_u. Za razliko od faze izločanja, usmerjanje pošlje pridobljen rezultat $data_p$, sestavljen iz besed $words_p = \{u_1, \ldots, u_v\}$ v iskalni kriterij s_u v primeru, če velja:

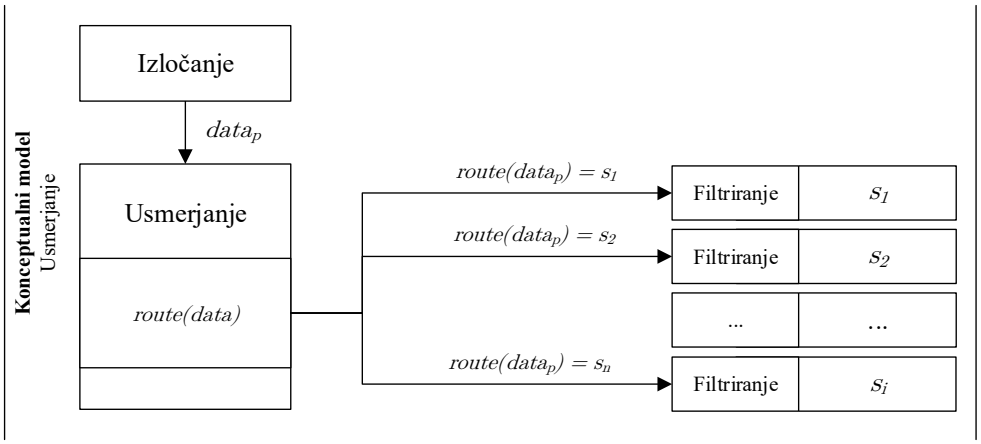
$$\exists\ w \in words_p \colon w \in W_{S_u} \tag{2.25}$$

Skladno s tabelo 3.1 je faza usmerjanja neizgubna, zato usmerjevalna funkcija vedno najde ciljni iskalni kriterij.

Usmerjevalna funkcija ***route*** je torej naslednja:

$$route(data_p) = s_u, \text{ kjer } \exists!\ \ s_u \in S, \exists\ w \in words_p \colon w \in W_{s_u} \tag{2.26}$$

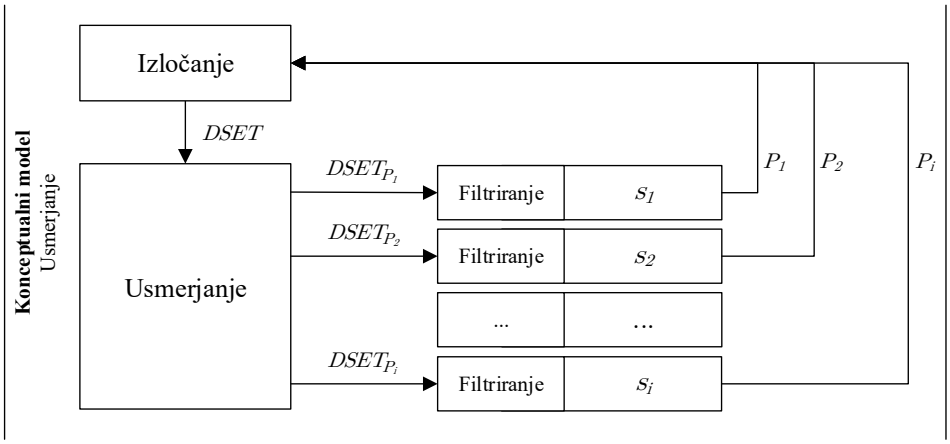
Izhod usmerjevalne funkcije je iskalni kriterij s_u v katerega sistem pošlje pridobljeni rezultat $data_p$. Spodnja shema prikazuje fazo usmerjanja za konstantni tok podatkov.



Slika 2.9: Faza usmerjanja za konstantni tok podatkov

2.3.2.2 Usmerjanje povpraševalnega toka

Usmerjanje povpraševalnega toka je trivialno, saj se ta generira na podlagi znanih povpraševanj (glej poglavje 2.2.2), za katere je vedno mogoče usmeriti rezultate povpraševanja v inicatorski iskalni kriterij.



Slika 2.10: Faza usmerjanja za povpraševalni tok podatkov

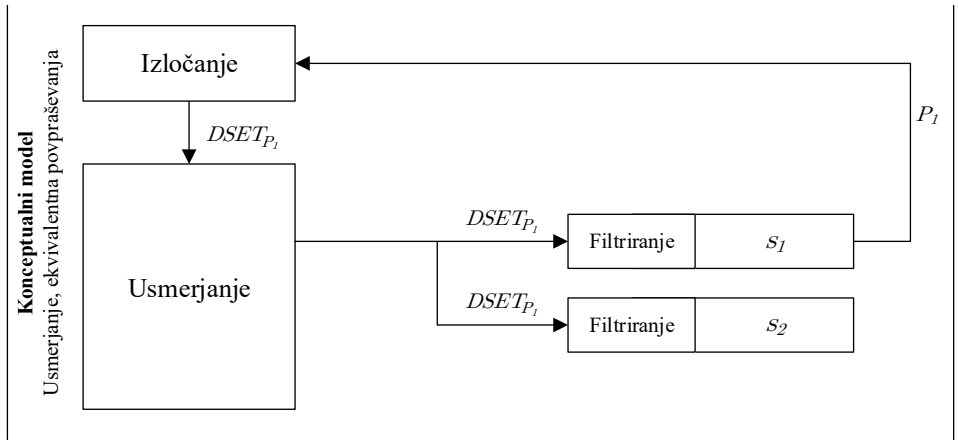
Ena od predvidenih optimizacij modela je, da se v primeru identičnih iskanj rezultati usmerijo na vse ekvivalentne iskalne kriterije.

Naj bo $DSET_{P_k} = (data_1, data_2, \ldots, data_m)$ tok povpraševanja P_k. V primeru, da obstaja več ekvivalentnih iskalnih kriterijev, potem velja:

$$x \neq k \;\exists\; P_x\colon P_x = P_k \text{ in } DSET_{P_k} = DSET_{P_x} \tag{2.27}$$

V takem primeru usmerimo vse pridobljene rezultate povpraševanja P_k na vsa ekvivalentna povpraševanja P_x, skladno z enačbo (2.27).

Naslednja shema prikazuje opisano optimizacijo:



Slika 2.11: Usmerjanje v primeru ekvivalentnih povpraševanj

2.3.3 Faza filtriranja

Faza filtriranja omogoča dodatno čiščenje podatkov glede na nastavitve filtrov znotraj iskalnih kriterijev. Namen filtriranja je v specifičnih zahtevah po izločitvi določene množice podatkov v iskalnem kriteriju, ki ustreza kriterijem filtrov[9].

Rezultati, ki so prispeli v fazo filtriranja so preživeli izločanje (in usmerjanje), kar pomeni, da so potencialno namenjeni shranjevanju in analiziranju v primeru, če jih faza filtriranja ne odstrani.

Definicija: Filtrirani podatki

Vsled nedvoumnosti definiramo filtrirane podatke kot podatke, ki preživijo fazo filtriranja in se uvrstijo v filtrirani podatkovni tok.

Definicija: Neželeni podatki

Analogno so neželeni podatki tisti, ki ne preživijo faze filtriranja in se uvrstijo v neželeni podatkovni tok. Privzeto se ti podatki zavržejo.

Naj bo $DSET$ podatkovni tok[10], ki prihaja v fazo filtriranja:

$$DSET = (data_1, data_2, \ldots, data_m) \tag{2.28}$$

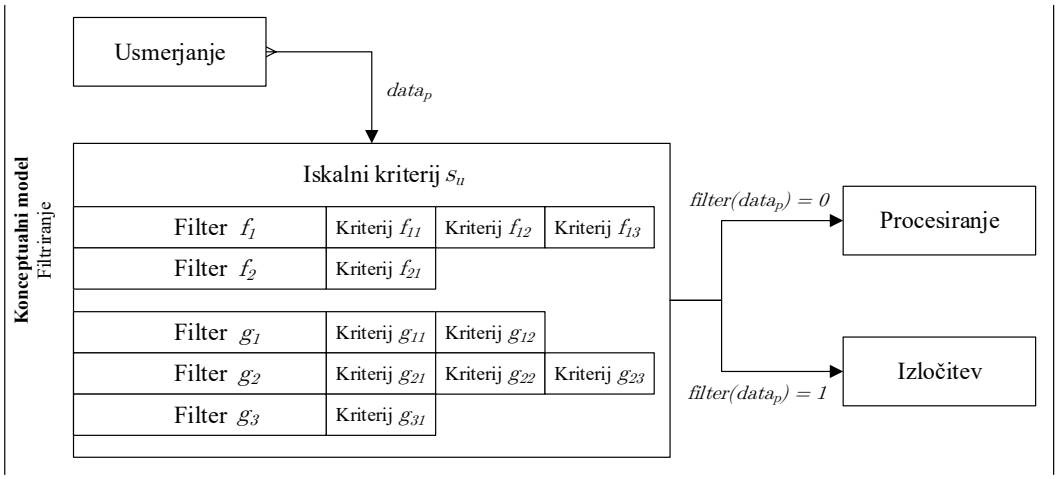
[9] Tipičen primer je izločitev vseh mnenj določenega uporabnika ali mnenj, ki vključujejo določeno besedo ali besedno zvezo.

[10] V fazi filtriranja sta podatkovna tokova $DSET_{T_k}$ in $DSET_{P_k}$ v celoti abstrahirana.

Naj bo $data_p$ p-ti rezultat v podatkovnem toku, skladno z enačbo (2.1), in naj bosta F in G množici filtrov, ki so aplikativni za rezultat $data_p$. Aplikativnost filtrov določa iskalni kriterij s_u, ki ga je določila faza usmerjanja. Množici F in G sta definirani kot:

$$F = \{f_1, \ldots, f_n\} \text{ in } G = \{g_1, \ldots, g_m\} \tag{2.29}$$

Kjer je f_i, za $1 \leq i \leq n$, posamezni *vključujoči filter* in g_j, za $1 \leq j \leq m$, posamezni *izključujoči filter* iskalnega kriterija s_u.



Slika 2.12: Faza filtriranja

Definicija: Vključujoči filter[11]

Vključujoči filter ***ne izloči*** *rezultatov, ki uspešno izpolnijo vsaj en kriterij filtra. Ostale rezultate filter izloči.*

Definicija: Izključujoči filter[12]

Izključujoči filter ***izloči*** *rezultate, ki uspešno izpolnijo vsaj en kriterij filtra. Ostalih rezultatov filter ne izloči.*

Tako vključujoči filter f_i kot izključujoči filter g_j imata lahko več kriterijev. Naj bodo f_{i_k} kriteriji filtra f_i za vsak k, kjer $1 \leq k \leq o$ in g_{j_l} kriteriji filtra

[11] Tipičen primer vključujočega filtra je *jezikovni filter*, ki določa kateri zaznani jeziki (na primer z ISO 639-1 kodami *si*, *en*, *de*) so dovoljeni. Faza filtriranja izloči vse rezultate, ki ne ustrezajo tem jezikom.

[12] Tipičen primer izključujočega filtra je *črkovni filter*, ki določa kateri znaki Unicode (na primer コ, ン, ピ, ュ, ー, タ) niso dovoljeni. Faza filtriranja izloči vse rezultate, ki vključujejo katerikoli tak znak.

g_j za vsak l, kjer $1 \leq l \leq p$. Kriteriji vključujočega filtra f_i vplivajo na rezultat filtra na sledeči način:

$$f_i(data_p) = f_{i_1} \vee f_{i_2} \vee ... \vee f_{i_o} \tag{2.30}$$

Izključujoči filtri so bolj restriktivni in je njihova funkcija z gledišča kriterijev naslednja:

$$g_j(data_p) = g_{j_1} \wedge g_{j_2} \wedge ... \wedge g_{j_p} \tag{2.31}$$

Torej za kriterije in filtre $f_1, ..., f_n, g_1, ..., g_m$ velja, da so definirani kot funkcije tako, da $\forall i \in \{1, ..., n\}$, $\forall j \in \{1, ..., m\}$, $\forall k \in \{1, ..., o\}$, $\forall l \in \{1, ..., p\}$ velja:

$$f_i\colon \{data_p\} \to \{0, 1\} \text{ in } g_j\colon \{data_p\} \to \{0, 1\} \tag{2.32}$$

$$f_{i_k}\colon \{data_p\} \to \{0, 1\} \text{ in } g_{j_l}\colon \{data_p\} \to \{0, 1\} \tag{2.33}$$

Ker filtri izločajo rezultate, vrednost 0 funkcij f_i in g_j pomeni ohranitev rezultata $data_p$ in 1 izločitev.

Naj bo $filter_V$ filtrirna funkcija *vključujočih filtrov*, ki je definirana kot:

$$filter_V(data_p) = \begin{cases} 0;\ \exists\ f_{i;\ i \in \{1,...,n\}}(data_p)\colon f_i(data_p) = 0 \\ 1;\ \forall\ f_{i;\ i \in \{1,...,n\}}(data_p)\colon f_i(data_p) = 1 \end{cases} \tag{2.34}$$

Naj bo $filter_I$ filtrirna funkcija *izključujočih filtrov*, ki je definirana kot:

$$filter_I(data_p) = \begin{cases} 0;\ \forall\ g_{j;\ j \in \{1,...,m\}}(data_p)\colon g_j(data_p) = 0 \\ 1;\ \exists\ g_{j;\ j \in \{1,...,m\}}(data_p)\colon g_j(data_p) = 1 \end{cases} \tag{2.35}$$

Filtrirna funkcija ***filter*** je torej naslednja:

$$filter(data_p) = filter_V(data_p)\ \wedge\ filter_I(data_p) \tag{2.36}$$

Naj bo $P(F)$ verjetnost uvrstitve rezultata v filtrirani podatkovni tok[13] in $P(N)$ verjetnost uvrstitve podatka v neželeni podatkovni tok. Ker sta to edina možna dogodka tudi tukaj velja:

$$P(F) = 1 \quad P(N) \tag{2.37}$$

V večini *praktičnih primerov* velja tudi:

$$P(F) \gg P(N) \tag{2.38}$$

[13] Filtrirani podatkovni tok predstavlja osnovo za kasnejše analize.

Enačba (2.38) pravi, da je odstotek zavrženih rezultatov v fazi filtriranja relativno majhen, saj je to zadnja faza v procesnem cevovodu. Večina rezultatov, ki pridejo v to fazo nadaljuje pot v analizo v obliki uporabnih podatkov, skladno z njihovo definicijo v poglavju 2.1.

2.3.3.1 Možnost hrambe filtriranih podatkov

Neželeni podatkovni tok vsebuje vse podatke, ki niso preživeli faze filtriranja in kršijo definicije filtrov kriterija v katerega so usmerjeni.

V modelu se predvideva hranjenje neželenih podatkov predvsem zaradi možnosti kasnejših analiz.

Ta način omogoča tudi naknadno vključevanje že izločenih podatkov v rezultate v primerih napak pri definiciji filtrov.

2.4 Model porazdeljevanja bremena

V tem poglavju podajamo konceptualni model za porazdeljevanje računskega bremena[14], ki omogoča porazdeljevanje na več procesnih enot. Način implementacije podajamo v poglavju 6.

Osnovni cilji modela so:

(1) Zagotoviti možnost visoke skalabilnosti procesiranja
(2) Podpreti poljubne vrste analiz nad zbranimi podatki
(3) Zagotoviti odpornost na napake
(4) Zagotoviti analize v kvazi realnem času

Ad (1) smo izvedli z implementacijo porazdeljenega procesiranja, ki temelji na mehanizmih nadrejenih-podrejenih instanc (master-slave[15]) in omogoča sodelovanje več enakovrednih računskih enot pri izvajanju različnih operacij / analiz.

---

[14] Imenujemo ga tudi model procesiranja. Vzrok za to je predvsem v dejstvu, da ima model samo dve fazi – fazo izbire instance nadrejene vloge in fazo procesiranja, pri čemer je druga bistvenega pomena.

[15] Gospodar-suženj. Zaradi premajhne uporabe tega izraza v slovenski tehnični stroki in prevelike konotacije na zgodovinske družbene ureditve bolj uveljavljenem slovenskem izrazu.

Ad (2) omogoča izvajanje poljubne računske operacije – analize iz nabora definiranih, na kateri koli procesni enoti[16]. Vrsta analize je predstavljena v glavi definicije dela (workload header) in omogoča porazdeljevanje vseh analiz na vse procesne enote.

> *Definicija: Analiza*
>
> *Analiza je računska operacija ali algoritem, ki teče nad filtriranim podatkovnim tokom iskalnega kriterija in ga lahko izvede poljubni procesor. Vsaka analiza je določene vrste.*
>
> *Definicija: Vrsta analize*
>
> *Vrsta analize določa algoritem, ki je uporabljen pri izvedbi analize. Nekatere analize se izvajajo v kvazi realnem času, druge samo na zahtevo.*

Ad (3) zagotavljamo zaradi arhitekture sistema za porazdeljevanje bremena in idempotentnosti analiz. Odpovedi ali napake procesnih enot bistveno ne vplivajo na delovanje celotnega sistema, saj je izračunavanje analiz izvedeno transakcijsko. V primeru izpada procesne enote analizo izvede naslednja procesna enota.

Ad (4) zagotavljamo z zmožnostjo samodejnega povečevanja virov v primerih, ko se zahtevki za procesiranje povečajo. Podrobnejši opis implementacije je podan v poglavju 6.

Model predvideva uporabo mehanizma nadrejenih-podrejenih instanc predvsem zaradi naslednjih zahtev:

(1) Delo naj razdeljuje ena procesna enota
(2) Neparalelizabilne operacije naj izvaja ena procesna enota
(3) Napaka ene ali več procesnih enot naj ne vpliva na stabilnost sistema

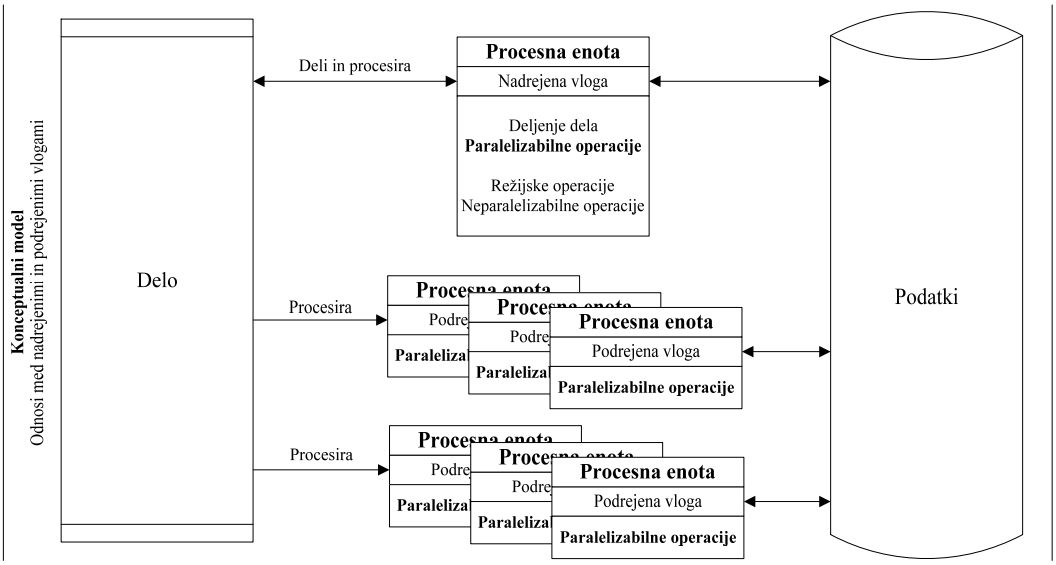
Razviti model predvideva uporabo ene (1) nadrejene procesne enote, kar v primeru neobvladovanja predstavlja enotno točko izpada. V modelu smo zato razvili način reševanja takih situacij s samodejno sposobnostjo hitre reelekcije (ponovnega voljenja) nadrejene instance v primerih, ko pride do izpada.

Poleg tega ima model porazdeljevanja nekaj ključnih lastnosti:

[16] Procesno enoto imenujemo tudi procesor.

(1) Vsaka procesna enota lahko opravlja vlogo nadrejene ali podrejene instance
(2) Vloga nadrejene instance se voli
(3) Ena nadrejena instanca vloge mora biti vedno aktivna
(4) Nadrejena vloga opravlja deljenje dela in procesira
(5) Nadrejena vloga opravlja režijsko delo in izvaja neparalelizabilne operacije
(6) Ena instanca podrejene vloge mora biti vedno aktivna
(7) Podrejena vloga izključno procesira

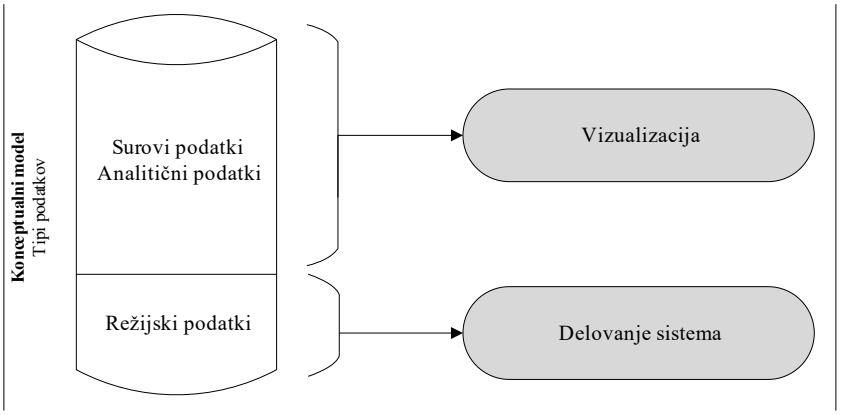
Spodnja shema prikazuje odnose med nadrejenimi in podrejenimi vlogami.



Slika 2.13: Odnosi med nadrejenimi in podrejenimi vlogami

S stališča podatkov namenjenih vizualizaciji, te ločimo na dva osnovna tipa: *surove podatke rezultatov* in *analitične podatke*. Hkrati v modelu hranimo tudi podatke, ki so pomembni za delovanje sistema[17] in jih imenujemo *režijski podatki*. Režijski podatki niso neposredno povezani z vizualizacijo in tudi niso porazdeljeni[18].

[17] Uporabniški podatki, nastavitve sistema, definicije iskalnih kriterijev, avtentikacijski podatki za dostopnike, itd.
[18] Porazdeljevanje podatkov se izvaja za surove podatke rezultatov in analitične podatke.



Slika 2.14: Surovi, analitični in režijski podatki

Surovi podatki so posledica filtriranega podatkovnega toka, ki je definiran v modelu izločanja, usmerjanja in filtriranja podatkov (točka (3), poglavje 2.3). Analitični podatki so izvedeni iz surovih podatkov na podlagi analiz, ki jih izvaja model porazdeljevanja bremena in so shranjeni dodatno. Model predvideva uporabo drsečega časovnega okna za definicijo roka hrambe tako za surove rezultate, kot analitične podatke.

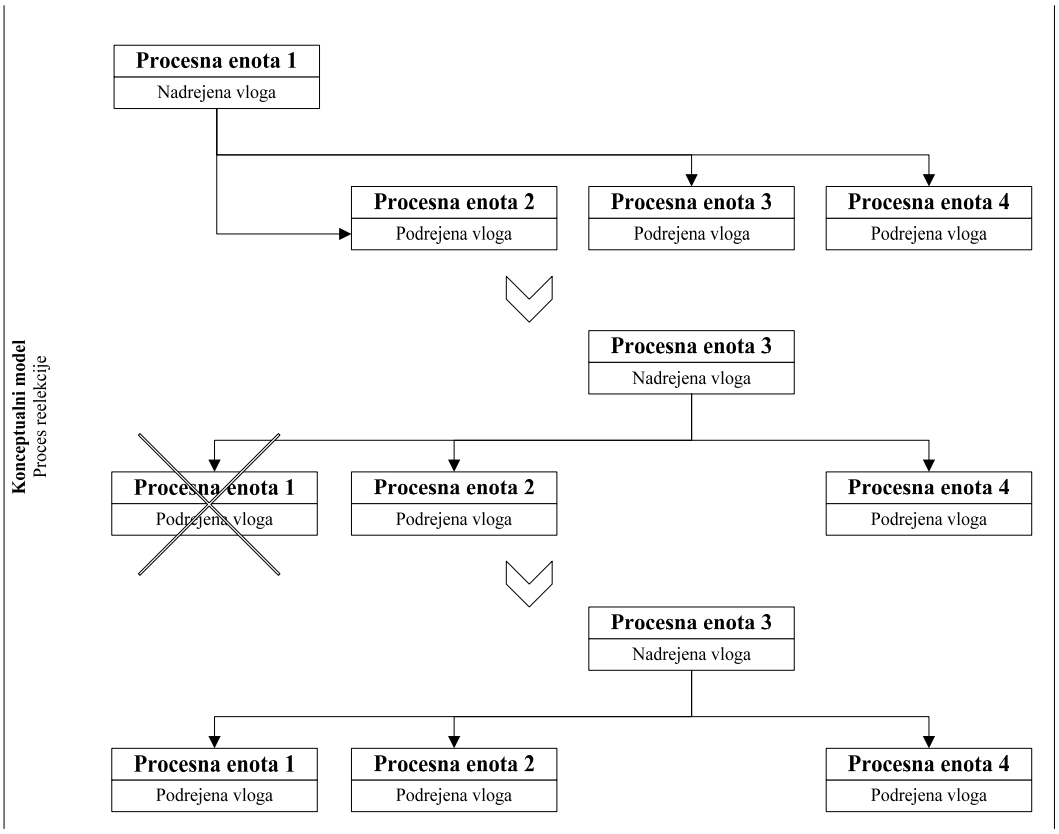
2.4.1 Faza elekcije

Model porazdeljevanja bremena glavno instanco (nadrejeno instanco) izbere z elekcijo. Vse instance se zavedajo svoje vloge in periodično preverjajo obstoj nadrejene instance. V primeru, če nadrejena instanca ne potrdi svojega delovanja v predvidenem času, se zgodi reelekcija.

Shema 2.15 prikazuje proces reelekcije, ki se zgodi, ko nadrejena instanca ne potrdi delovanja v predvidenem času. V takem primeru se zgodi reelekcija nove nadrejene instance iz bazena delujočih podrejenih instanc, nadrejeno instanco pa se odstrani iz nabora aktivnih instanc in reciklira.

Naj bo $P = \{p_1, p_2, \dots, p_n\}$ množica n procesnih enot, $p_{m;\, m \in \{1,\dots n\}}$ nadrejena instanca in e_p perioda preverjanja procesnih enot za obstoj nadrejene instance. Ker se instance zaženejo med seboj neodvisno lahko predpostavimo, da njihove periode niso časovno sinhronizirane, pač pa je povprečna perioda $\bar{e}$ med povpraševanji (s strani katere koli instance) omejena z:

$$0 \leq \bar{e} \leq e_p \tag{2.39}$$



Slika 2.15: Proces reelekcije

V primeru enakomerne porazdelitve, ki je s stališča zagotavljanja obstoja nadrejene instance najvarnejša, imamo minimalno periodo preverjanja definirano z:

$$e_{min} \geq \frac{e_p}{n} \tag{2.40}$$

Tipična perioda preverjanja e_p je 1000 milisekund. V primeru 16 procesnih enot to pomeni, da je povprečen čas med preverjanji za obstoj nadrejene instance enak 62,5 milisekunde. V primeru njene odpovedi bi celoten sistem torej ostal brez nadrejene instance in posledično brez dodeljevanja novega dela največ 62,5 milisekund.

V primeru, ko katera koli instanca ob času preverjanja ne dobi odgovora o obstoju nadrejene instance se sproži *reelekcija*. Ta poskrbi, da se izvoli nova nadrejena instanca.

Reelekcijski algoritem ni predmet konceptualnega modela in je opisan v poglavju 6.1.

2.4.2 Faza procesiranja

Faza procesiranja je namenjena generiranju analiz iz podatkov filtriranega podatkovnega toka. Model generira analizo za vsako vrsto analize, za vsako časovno okno in za vsak iskalni kriterij.

Definicija: Časovno okno

Časovno okno je obdobje v času za katero se analiza izračunava. Model predvideva dva tipa oken – fiksno in nastavljivo. Fiksna časovna okna so dolga: 1 minuto, 3 minute, 5 minut, 10 minut, 15 minut, 30 minut, 1 uro, 2 uri, 3 ure, 6 ur, 12 ur, 1 dan, 2 dni in 7 dni. Nastavljivo časovno okno določata dve točki v času, t_a in t_b, kjer je $t_a < t_b$.

Konceptualni model
Vrste analiz in časovna okna

Analiza števila zadetkov	1 min	3 min	5 min	10 min	15 min	30 min	1 h	2 h	3 h	6 h	12 h	1 dan	2 dni	7 dni
Analiza dosega	1 min	3 min	5 min	10 min	15 min	30 min	1 h	2 h	3 h	6 h	12 h	1 dan	2 dni	7 dni
Analiza absolutnega sentimenta	1 min	3 min	5 min	10 min	15 min	30 min	1 h	2 h	3 h	6 h	12 h	1 dan	2 dni	7 dni
Analiza relativnega sentimenta	1 min	3 min	5 min	10 min	15 min	30 min	1 h	2 h	3 h	6 h	12 h	1 dan	2 dni	7 dni
Analiza besednega oblaka	2016-06-01 – 2016-06-03				2016-06-06 13:00:00 – 2016-06-08 14:30:00						2016-07-12 – 2016-07-14			
...	...													

Slika 2.16: Vrste analiz in časovna okna

Zgornja shema prikazuje primer z nekaj vrstami analiz in fiksnimi časovnimi okni ter primer analize besednega oblaka z nastavljivimi časovnimi okni, ki jih zahteva uporabnik. Faza procesiranja generira vse analize fiksnih časovnih oken enkrat na periodo generiranja analiz. Nastavljiva časovna okna se analizirajo takoj po vpisu zahtevka s strani uporabnika oziroma ob naslednji periodi.

Definicija: Perioda generiranja analiz

Perioda generiranja analiz je čas med dvema razdeljevanjema dela. Ob času periode generiranja analiz, nadrejena instanca razdeli delo vsem podrejenim instancam. Perioda generiranja analiz je podana v minutah (min).

Naj bo w število časovnih oken, t število vrst analiz, p perioda generiranja analiz in s število iskalnih kriterijev. Ker se v fazi procesiranja za vsak iskalni kriterij generirajo vse podrte vrste analiz za vsa časovna okna, je število analiz N_p za katere je potrebno generirati definicije (razdeliti delo) in jih izračunati (opraviti delo) v časovni periodi p torej enako:

$$N_p = w \times t \times s \tag{2.41}$$

Ena od lastnosti iskalnih kriterijev je tudi prioriteta izračunavanja analiz. Kriteriji z višjo prioriteto imajo krajšo periodo p za izračun analiz. Naj bodo $\{p, p_2, \ldots, p_n)$ analitične časovne periode, ki ustrezajo prioritetam. Hitrost izračunavanja analiz na minuto, za periodo p, je torej:

$$v = \frac{N_p}{p} = \frac{w \times t \times s}{p} \ (^{analiz}/_{minuto}) \tag{2.42}$$

Povprečna zahtevana hitrost analiz čez vse prioritete je definirana kot:

$$\bar{v} = \frac{\sum_{p_i} (\frac{N_{p_i}}{p_i})}{n} \ (^{analiz}/_{minuto}) \tag{2.43}$$

Definicija: Obvladljivost procesiranja analiz

*Procesiranje analiz je **obvladljivo** v primeru, ko je povprečna hitrost analiz manjša ali enaka sposobnosti procesiranja vseh procesnih enot. V primeru, da je sposobnost procesiranja manjša od povprečne hitrosti analiz se zahtevki za analize kopičijo v vrsti in jih je potrebno izvesti kasneje, z več procesnimi enotami.*

2.4.3 Primer števila in hitrosti izračunavanja analiz

Podajamo realen primer izračuna števila analiz in zahtevane hitrosti izračunavanja analiz.

Podatki za izračun:

- Število iskalnih kriterijev: 500
  - 150 prioritete 1: perioda analiz: 1 minuta
  - 50 prioritete 2: perioda analiz: 5 minut
  - 25 prioritete 3: perioda analiz: 15 minut
  - 100 prioritete 4: perioda analiz: 30 minut
  - 175 prioritete 5: perioda analiz: 60 minut
- Število časovnih oken: 14
- Število vrst analiz: 12

Število analiz prioritete 1: $N_{p_1} = w \times t \times s = 14 \times 12 \times 150 = 25.200$

Hitrost analiz prioritete 1: $v_{p_1} = \frac{N_{p_1}}{p_1} = \frac{25.200}{1} = 25.200/minuto$

Število analiz prioritete 2: $N_{p_2} = 8.400$
Hitrost analiz prioritete 2: $v_{p_2} = 1.680/minuto$

Število analiz prioritete 3: $N_{p_3} = 4.200$
Hitrost analiz prioritete 3: $v_{p_3} = 280/minuto$

Število analiz prioritete 4: $N_{p_4} = 16.800$
Hitrost analiz prioritete 4: $v_{p_4} = 560/minuto$

Število analiz prioritete 5: $N_{p_5} = 29.400$
Hitrost analiz prioritete 5: $v_{p_5} = 490/minuto$

Povprečna zahtevana hitrost analiz:

$$\bar{v} = \frac{25.200 + 1.680 + 280 + 560 + 490}{5} = \frac{5.642}{minuto} \quad \frac{94}{sekundo}$$

V pričujočem primeru se torej zahteva konstantna hitrost, ki je večja od *94 analiz vsako sekundo*, oziroma eno analizo na vsakih *10,6 milisekund*.

2.5 Model vizualizacije analitičnih podatkov

V tem poglavju podajamo konceptualni model vizualizacije analitičnih podatkov, ki je namenjen *prikazu rezultatov analiz* končnim uporabnikom. S stališča uporabnika obstajajo trije načini za prikaz analiz, in sicer:

(1) Prikaz predizračunanih (precalculated) analiz
Predizračunane analize sistem izračunava vnaprej z namenom boljše uporabniške izkušnje ob priklicu analize. Temeljijo na fiksnih časovnih oknih in jih uporabniki največkrat uporabljajo pri analiziranju problemske domene.

(2) Izračun analiz na zahtevo in takojšen prikaz
Določene vrste analiz niso primerne za izračunavanje vnaprej, saj so smiselne samo pri večji količini podatkov in bi bilo vnaprejšnje izračunavanje izjemno podatkovno potratno. Uporabniki zato samostojno izberejo časovno obdobje in tip analize. Analiza se izračuna takoj po oddani zahtevi.

(3) Izračun analiz na zahtevo in zakasnjen prikaz
Določene vrste analiz dovoljujejo zakasnjeni prikaz v primerih, ko se uporabnik odloči za izvedbo analize, katere izračun lahko traja nesprejemljivo dolgo. Uporabnik odda naročilo za analizo in se vrne po

končanem izračunu. Ta vrsta analiz je aplikativna predvsem v primeru daljših časovnih obdobij, ki posledično pomenijo tudi več vhodnih podatkov za analizo.

V naslednjih poglavjih podajamo nekaj primerov osnov vizualizacije za grafikone, ki so najbolj uporabni za prikaz različnih vrst analiz.

2.5.1 Črtni grafikoni

Črtni grafikoni prikazujejo eno ali več lastnosti[19] iskalnega kriterija na dveh oseh. V večini primerov govorimo o vpogledu v količine, ki se spreminjajo skozi čas.

Tipični primeri črtnih grafikonov uporabljenih za reševanje domene tekstovnih zbirk (ali tekstovnih objav) so:

(1) Število zadetkov
Ta vrsta analize določa število zadetkov z določenim časovnim oknom v določenem časovnem obdobju.

(2) Doseg
Analiza dosega določa potencialni maksimalni doseg analiziranih vsebin na podlagi števila sledilcev / prijateljev.

(3) Sentiment
Analiza sentimenta določa absolutni sentiment in relativni sentiment analiziranih vsebin skozi čas.

Naj bo s iskalni kriterij in t časovno obdobje za katerega želimo prikazati črtni grafikon. Časovno obdobje je definirano na polodprtem intervalu, v sekundah, kot:

$$t = [t_a, t_b) \tag{2.44}$$

Naj bo perioda analiz za kriterij s enaka p in enako definirana v sekundah. Naj bo N_r število rezultatov in N_l število lastnosti, ki jih želimo prikazati uporabniku. Potem imamo na intervalu t naslednje število rezultatov analiz:

$$N_r = \frac{t_b \quad t_a}{p} \times N_l \tag{2.45}$$

[19] Tipično je to številska vrednost, ki prikazuje (na primer) količino zadetkov, količino sledilcev ali število pozitivno/negativno/nevtralno konotiranih rezultatov.

V primeru prikaza analize sentimenta, kjer prikazujemo tri lastnosti (število najdenih pozitivnih sentimentov, število najdenih negativnih sentimentov, absolutna razlika) je za naslednja obdobja potrebno iz podatkovne baze, ob predpostavki, da se analize generirajo enkrat na minuto, pridobiti naslednje število analiz – vrstic v podatkovni bazi in jih vizualizirati uporabniku:

$$N_{r_{sentiment\_hour}} = \frac{60 \times 60}{60} \times 3 = 180 \tag{2.46}$$

$$N_{r_{sentiment\_day}} = \frac{24 \times 60 \times 60}{60} \times 3 = 4.320 \tag{2.47}$$

$$N_{r_{sentiment\_week}} = \frac{24 \times 60 \times 60 \times 7}{60} \times 3 = 30.240 \tag{2.48}$$

$$N_{r_{sentiment\_month}} = \frac{24 \times 60 \times 60 \times 30}{60} \times 3 = 129.600 \tag{2.49}$$

2.5.2 Besedni oblaki

Besedni oblaki označujejo frekvenco uporabljenih besed (unigramov) ali besednih zvez (bigramov, trigramov, ...). Besedni oblaki so lahko kombinirani z dodatnimi lastnostmi in z barvo odražajo drugo lastnost – na primer sentiment.

Tipični primeri vrst analiz, ki za prikaz uporabljajo besedni oblak so:

(1) Besedni oblak rezultatov
   Besedni oblak prikazuje katera beseda ali besedna zveza je bila v določenem časovnem obdobju največkrat uporabljena in kakšen je njen sentiment v kontekstu celotne objave.

(2) Besedni oblak značk
   Oblak značk prikazuje katere značke so bile največkrat uporabljene pri spremljanju iskalnega kriterija skozi čas.

(3) Komunikacijski oblak
   Oblak komunikacije odraža sodelujoče uporabnike pri iskalnem kriteriju neke téme.

(4) Percepcijska analiza
   Percepcijska analiza prikazuje unigrame, bigrame ali trigrame na dvodimenzionalni percepcijski ravnini, kjer ena dimenzija pomeni frekvenco, druga pa sentiment. Osnovni podatki za to analizo so pridobljeni iz podatkov besednih oblakov.

Več o izvedbi modela vizualizacije za različne vrste analiz podajamo v poglavju 7.1.

3 Problemska domena

Predlagani konceptualni model za porazdeljeno procesiranje, analizo in vizualizacijo podatkov z mehanizmi visoke skalabilnosti smo testirali na specifični problemski domeni, ki zadošča naslednjim zahtevam:

- **Dinamična domena**
  Podatki uporabljene domene so dinamični in hitro spremenljivi. Spremembo podatkov je potrebno upoštevati hitro in s čim manjšo časovno zakasnitvijo.
- **Velika količina podatkov**
  Domena vključuje zajem velike količine podatkov. Količina shranjenih podatkov je seveda odvisna od števila iskalnih kriterijev in nastavitev sistema za hrambo, hkrati pa se skozi sistem prenaša praktično enaka količina podatkov, ne glede na čas.
- **Širok nabor zahtevanih analiz**
  Problemska domena vključuje širok nabor zahtevanih analiz, ki se izvajajo nad istimi izvornimi podatki.
- **Heterogenost analiz**
  Zahtevane analize so zelo heterogene, tako po kompleksnosti in trajanju izračuna, kot tudi po informacijskem doprinosu. Nekatere analize so v vseh pogledih trivialne (npr. štetje zadetkov), medtem, ko druge analize omogočajo poglobljen pogled v trenutno stanje domene.
- **Napredne vizualizacije**
  Dotična problemska domena za intuitivno predstavitev rezultatov analiz zahteva napredne vizualizacije in uporabniške vmesnike, kot

so besedni oblaki, intuitivni časovni selektorji, hitre povečevalne funkcije, označevanje grafov, več-črtni grafikoni, ploščinski grafikoni, specifični grafikoni sentimenta in korespondenčni zemljevidi.

3.1 Opis domene

Problemska domena, na kateri smo v magistrskem delu testirali model porazdeljevanja bremena, je ugotavljanje širokega mnenja javnosti[20] glede poljubne téme, osebe, blagovne znamke ali drugega družbeno pomembnega dogodka ali dejavnika o katerem je mogoče pridobiti avtenticirana mnenja[21] preko dostopnih, javnih informacij na internetu.

Definicija: Široko mnenje javnosti

V tem delu, široko mnenje javnosti definiramo kot nabor vseh najdenih vsebin pri ponudnikih na internetu, ki ustreza kriterijem zajema glede na iskalni kriterij.

Definicija: Iskalni kriterij

Iskalni kriterij določa kriterije iskanja vsebin, ki določajo specifično témo, področje, osebo, blagovno znamko, ipd. in omejitve ter potencialne izločilne kriterije - filtre.

S stališča pogleda na široko mnenje javnosti je potrebno opozoriti, da tega ne istovetimo z javnim mnenjem, ki predstavlja specifično določen družboslovni pojem, pač pa mnenje širokega nabora različnih ljudi. Z izbiro omrežij s katerih pridobivamo podatke smo povečali tudi reprezentativnost mnenj, katere lahko za občutljive téme dodatno filtriramo s pametnimi filtri, ki so opisani v naslednjih poglavjih.

V splošnem smo za testiranje modela potrebovali problemsko domeno, ki vključuje veliko količino podatkov in ima zahtevo po konstantnem in hitrem spreminjanju količine, vsebine in dinamike podatkov, saj smo le tako lahko testirali kvaliteto in odzivnost ključnih analiz nad podatkovnimi tokovi, ki jih s pomočjo problemske domene analiziramo.

---

[20] Široko mnenje javnosti ni ekvivalentno javnemu mnenju.

[21] Mnenja avtenticiranih, registriranih uporabnikov na družbenem omrežju Twitter ali Facebook.

Problemska domena odgovarja na naslednja vprašanja:

- **Kako se ljudje počutijo kadar govorijo o določeni témi[22]?** Pozitivno, negativno ali nevtralno in s kakšno zanesljivostjo lahko to napovemo?
- **Koliko ljudje govorijo o določeni témi?** Število zadetkov se preiskuje v različnih časovnih obdobjih in za različno velika drseča časovna okna.
- **Kakšen je doseg vsebin o določeni témi?** Koliko drugih ljudi je videlo sporočila o specifični témi, kjer se skupno število ljudi preiskuje za različna časovna obdobja in različno velika časovna okna.
- **Kakšna je resonanca ali odzvok o določeni témi?** Koliko ljudi je povzelo, potrdilo ali delilo specifično vsebino o določeni tmi?
- **Kdo so mnenjski vodje ali generatorji vsebin?** Kakšna je količina mnenj, glede na specifičnega uporabnika, o določeni témi?
- **Kdo so najbolj aktivni uporabniki?** Podaja vrstni red najbolj aktivnih (najbolj pišočih) uporabnikov glede na specifično témo.
- **Katere besede in besedne zveze se največkrat uporabljajo skupaj z analizirano témo?** Kakšni so besedni oblaki unigramov, bigramov ali trigramov?
- **Kakšne značke se najbolj uporabljajo v povezavi z analizirano témo?** Katere značke (hashtags) so največkrat uporabljene in s čem ljudje dodatno označujejo podano vsebino?
- **Kakšna je percepcijska (zaznavalna) analiza / korespondenčni zemljevid za določeno témo?** Kako se s stališča frekvence uporabljenih besed rezultati umeščajo na ravnino sentiment/frekvenca?

[22] Téma v širokem pomenu besede lahko vključuje ključne besede, značke ali drugače zastavljen kriterij.

- **Kakšna je komunikacijska mreža med uporabniki?** Kdo s kom največ komunicira? Kdo največ komunicira z drugimi uporabniki o določeni témi?
- **Kakšni so neposredni rezultati?** Prikaz surovih najdenih rezultatov, ki ustrezajo iskalnemu kriteriju in spoštujejo omejitve filtrov.

V opisu vprašanj problemske domene so izpostavljena ključna vprašanja, ki si jih zastavljamo pri reševanju problemske domene in na katera je mogoče delno ali v celoti odgovoriti na podlagi analiz podanih s strani konceptualnega modela izračunavanja podanega v poglavju 2.

3.2 Preslikava modela v problemsko domeno

Za uspešno preslikavo modela v problemsko domeno je potrebno:

(1) pridobiti čim bolj reprezentativne podatke
(2) podatke izločiti, če niso aplikativni
(3) podatke usmeriti na dotične iskalne kriterije
(4) podatke filtrirati skladno z nastavitvami filtrov
(5) podatke shraniti
(6) podatke analizirati
(7) podatke vizualizirati

Ad (1) v modelu rešujemo z izbiro izvornih ponudnikov široke komunikacije, specifično družbenih omrežij Twitter[23] in Facebook[24]. Model pridobivanja podatkov preko tokovnega vmesnika posluša celotno vsebino omrežja Twitter (Twitter firehose[25]) in dodatno povprašuje preko vmesnika API, ki ga podjetje ponuja javno. Hkrati model pridobiva ekvivalentne podatke iz javnih Facebook strani, ki so vnaprej vpisane kot potencialni izvor relevantnih informacij.

[23] http://www.twitter.com
[24] http://www.facebook.com
[25] https://dev.twitter.com/streaming/firehose. V obdobju od septembra 2014 do septembra 2015 je sistem preko tokovnega vmesnika zajel povprečno 4.900 tvitov na sekundo.

Ad (2) rešujemo v primerih, ko je na voljo širša množica podatkov / iskalnih rezultatov od želene. To je posebej pomembno v primeru obvladovanja konstantnih tokov (glej poglavje 2.2.1), kjer zavračamo veliko večino rezultatov.

Ad (3) rešujemo z modelom usmerjanja ujemajočih rezultatov v primeren iskalni kriterij z uporabo mehanizma založnik-naročnik (pub-sub) in visoko performančnega usmerjevalnika.

Ad (4) rešujemo z različnimi tipi filtrov, ki so del modela za filtriranje. Filtriranje se izvaja na več točkah in omogoča vključujoče in izključujoče filtre.

Ad (5) rešujemo s porazdeljeno relacijsko podatkovno bazo v oblaku, ki lahko shranjuje velike količine podatkov.

Ad (6) rešujemo z analitičnimi operacijami, ki jih model porazdeljevanja bremena lahko izvaja paralelno na več računskih enotah v oblaku.

Ad (7) prikazujemo z uporabo naprednih vizualnih komponent, ki uporabniku analitične informacije prikazujejo na intuitiven in vizualen način.

3.3 Javno komuniciranje

J. Bolen et. al. ugotavljajo, da je tvit "mikroskopska, časovno avtentična pojavitev sentimenta" [16]. Argumentirajo celo, da v obdobju netipičnih dogodkov, ko se pojavnost na tem omrežju poveča, zapisani tviti odražajo javno mnenje.

Raziskovanje javnega mnenja, niti pristopi k njemu ali algoritmi uporabljeni v eni od analiz predstavljenega modela niso predmet tega dela, vseeno pa je smiselno poudariti, da je v družboslovnih krogih ideja neintruzivnega[26] preučevanja javnega mnenja dobro preučena [17].

Stimson poudarja, da je realno mnenje vedno podano bolj iskreno v primerih, ko sogovornik ni aktivno izzvan z vprašanjem, pač pa samoiniciativno poda mnenje na preučevano témo.

Informacija pridobljena s pasivnim poslušanjem je, kadar je oddana s strani nekoga, ki skrbi za svoj ugled, kvalitetnejša in bolj istovetna oz. iskrena. Glede na to, da je pristop k pasivnem poslušanju v tem magistrskem delu

[26] Merjenje javnega mnenja brez aktivnega spraševanja korespondentov, samoizjava.

izveden nad avtenticiranimi profili uporabnikov na družbenih omrežjih, je v veliki večini primerov mogoče sklepati, da dotična mnenja odražajo realno pozicijo in razmišljanje avtorjev izjav.

Metrike javnega komuniciranja se nikakor ne omejujejo samo na čustvene (sentimentalne) odzive in podajajo veliko informacij tudi iz drugih tipov analiz. Količina pogovorov na primer podaja pomembnost določenega iskalnega kriterija za širšo javnost. Doseg (reach) vsebine prikazuje moč aktivacije javnosti, posebej v primerih uporabnikov, ki imajo velik krog sledilcev. Odstotek aktivacije mnenjskih voditeljev je posebna kategorija preiskovanja, ki se v analizah javnega mnenja obravnava z veliko pozornostjo. R. S. Burt argumentira [18], da so mnenjski voditelji posredniki med skupinami ljudi in torej opravljajo pomembno vlogo prenosa informacij med sicer disjunktnimi množicami uporabnikov družbenih omrežij. Mnenjski voditelji imajo pomemben vpliv na množico tém, oziroma lahko vplivajo na potencialno spremembo mnenja ljudi, ki voditelje spremljajo [19]. M. Cha et. al. so mnenjske voditelje identificirali na podlagi treh ključnih karakteristik omrežja Twitter: števila sledilcev, števila retvitov (retweets) in števila omemb. Odkrite karakteristike razlagajo, da število sledilcev ni faktor, ki najpomembneje doprinese k voditeljevem vplivu na njegove poslušalce, pač pa določa predvsem njegovo popularnost. Število omemb in ponovitev tvitov pa sta s stališča identifikacije mnenjskih voditeljev signifikantnejša [19].

Glede na naravo moderne komunikacije, ki se v širšem krogu, v času pisanja tega dela odvija ravno na spletu, E. Dubois ugotavlja, da klasična definicija mnenjskega voditelja drži tudi v primeru družbenih omrežij [20]. Skladno s tem prepričanjem drži tudi hipoteza dvostopenjskega toka komunikacije [21], kjer masovni mediji vplivajo na mnenjske voditelje, ti pa ideje razširjajo v splošno javnost.

Pristopi k analizi javnega komuniciranja se konceptualno torej bistveno ne spreminjajo zaradi prehoda komunikacije v elektronske medije. Predvsem se spreminja velikost problemske domene za analizo, saj je količina podatkov, usmerjanje, njihova analiza in vizualizacija za nekaj razredov večja od količine pri klasičnem pristopu merjenja javnega mnenja. Ugotavljanje širokega mnenja javnosti zato s klasičnim pristopom, ni ne vsebinsko, ne tehnično primerljiv postopek.

3.4 Način pristopa

Zaradi potrebe po legalnem pridobivanju najširšega možnega nabora pogovorov o določeni témi na samodejni način (z uporabo tehnologije, ne ročno) so družbena omrežja trenutno edini primeren kandidat. Količina informacij, ki je dosegljiva preko tega kanala je neprimerno večja, kot kateri koli drugi pristop, ki ima za cilj oblikovanje trendov širokega mnenja javnosti.

Pristop k analizi opisane problemske domene vsekakor ni striktno samodejni, pač pa omogoča ročno intervencijo v primerih, ko sociološka vedenja pripeljejo do absurdnih rezultatov analiz. Znotraj modela prikaza analiz je zato možna ročna intervencija, ki omogoča odpravo tipičnih pojavitev šuma. Nekaj izzivov, ki jih povzroča šum je podanih v naslednjem poglavju.

Glede na to, da je bila problemska domena izbrana zaradi potrebe po veliki količini podatkov in uporabnosti izvedenega dela, smo razvili skalabilen teoretični model za pridobivanje, procesiranje in analiziranje celotnega toka podatkov iz družbenih omrežij Twitter in Facebook[27] z možnostjo razširitve na druge izvore podatkov v prihodnosti.

3.5 Izzivi

Družbena omrežja vsekakor predstavljajo nereprezentativen vzorec [22]. Izkazuje se, da vzorec ni utežen tako pri starosti uporabnikov, kot izobrazbi uporabnikov, saj je pristranski in nagnjen k populaciji med 20 in 45 let, ki ima visokošolsko ali univerzitetno izobrazbo.

Hkrati družbena omrežja, podobno kot iskalniki, ustvarjajo kognitivni balon s klasičnimi homofilnimi dejstvi, da *prijatelj mojega prijatelja je moj prijatelj* in *iste ptice letijo skupaj*, kar družbena omrežja s pridom izkoriščajo s predlagalnimi algoritmi za sklepanje novih povezav [23].

Kompleksnost socialno-kulturnih interakcij in obnašanje uporabnikov, ki je namenjeno algoritmični nevidnosti (algorithmic invisibility) dodatno otežuje interpretacijo najdenih rezultatov. Uporaba sarkazma, slik namesto teksta,

[27] Skladno z omejitvami Facebook API.

podtvitanje (subtweeting), namerno ironično retvitanje (ironic retweets) pomembno otežujejo izdelavo kvalitetnih analiz in v določenih primerih zahtevajo ročno intervencijo. Ti primeri so posebno očitni pri iskalnih kriterijih, ki imajo manjše število rezultatov, saj lahko nekaj uporabnikov popolnoma pokvari realno sliko preostale večine.

4 Model pridobivanja podatkov

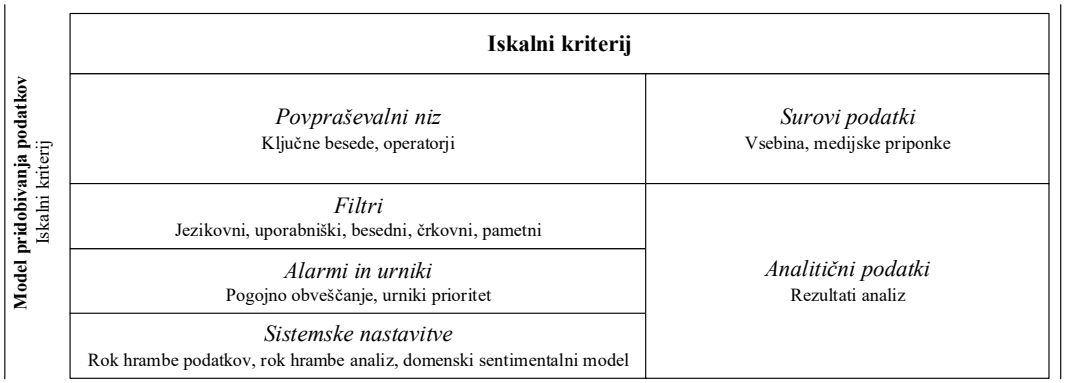
V poglavju podajamo implementacijo modela za pridobivanje podatkov, ki skrbi za pridobivanje, izločanje usmerjanje, filtriranje in shranjevanje podatkov, ki jih zahtevajo iskalni kriteriji.

4.1 O iskalnih kriterijih

Iskalni kriterij je sestavljen iz množice ključnih besed, Boolovih operatorjev, filtrov in drugih operatorjev, ki določajo načine iskanja po družbenih omrežjih Twitter in Facebook.

Kriteriji predstavljajo najmanjšo enoto iskanja, ki je lahko porazdeljena – v primeru izvedbe pridobivanja podatkov s povpraševanji (povpraševalni podatkovni tok, poglavje 2.2.2).

Spodnja shema prikazuje sestavo iskalnega kriterija, podatkov in analiz, ki mu pripadajo.

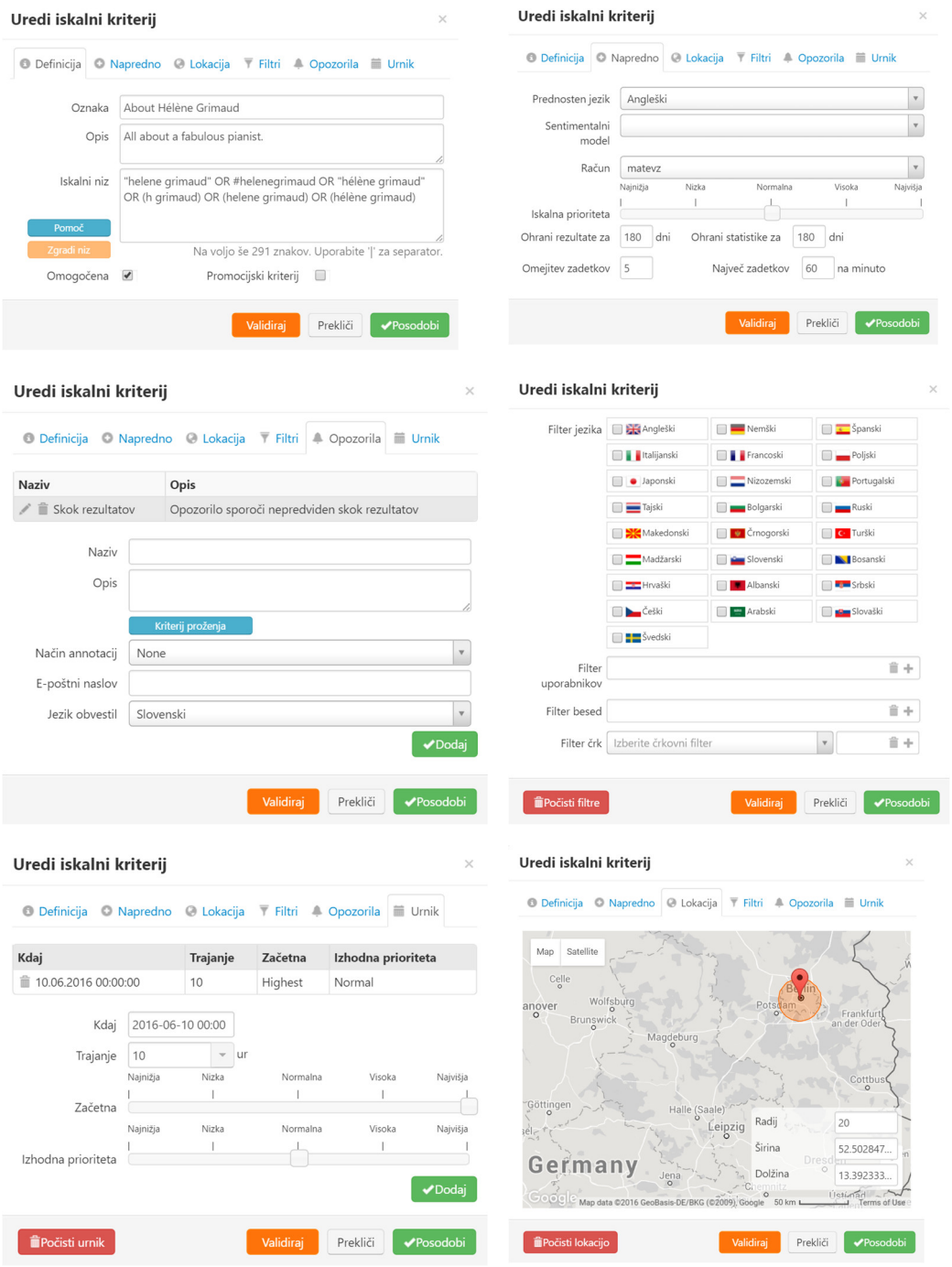


Slika 4.1: Shema iskalnega kriterija

Predstavitev iskalnega kriterija uporabniku omogoča nastavljanje vseh zgoraj prikazanih lastnosti preko intuitivnega uporabniškega vmesnika.

Uporabnikom se ni potrebno zavedati kompleksnosti delovanja rešitve in kriterije nastavljajo brez poznavanja delovanja sistema.

Spodnje slike prikazujejo uporabniški vmesnik za nastavljanje lastnosti iskalnega kriterija:



Slika 4.2: Uporabniški vmesnik za lastnosti kriterija

V naslednjih poglavjih podajamo podrobnejšo implementacijo posameznih lastnosti iskalnih kriterijev.

4.1.1 Ključne besede in operatorji iskalnih kriterijev

V povpraševalnem nizu[28] iskalnega kriterija so podprti Boolovi operatorji *in* (AND), *ali* (OR) in *asociativni operator*. Ostali napredni operatorji, ki so vezani na problemsko domeno, so podani v spodnji tabeli:

operator	**rezultat**
public perception analysis	vse besede, "public", "perception" in "analysis"
"positive vibrations"	točna fraza "positive vibrations"
plus OR minus	vsebuje besedo "plus" ali "minus" ali obe
#perception	vsebuje značko "#perception"
from:matevzg	poslano od osebe/računa "matevzg"
to:elonmusk	poslano osebi/računu "elonmusk"
@MIT	omenja osebo/račun "MIT"
germany AND geocode:"46.05223,14.50567,20km"	poslano v radiju 20 km od določenih koordinat in vsebuje besedo "germany"
politics since:2016-06-27	vsebuje besedo "politics" in poslano od datuma 2016-06-27
australia until:2016-06-22	vsebuje besedo "australia" in poslano do datuma 2012-06-22
weather ?	vsebuje besedo "weather" in uporablja vprašalno obliko
audi filter:links	vsebuje besedo "audi" in vključuje povezavo/e

Tabela 4.1: Napredni operatorji iskalnih kriterijev

V iskalnih kriterijih je podprto združevanje vseh operatorjev z asociativnim operatorjem. Tako je mogoče sestaviti poljubno kompleksne iskalne pogoje,

[28] Povpraševalni niz (query string) je celoten niz, ki je sintaktično pravilno oblikovan in ga sistem lahko razčleni in uspešno pošlje na aplikacijski vmesnik ponudnika podatkov.

ki vključujejo različne operatorje. Asociativni operator je posebno koristen v primerih, ko je potrebno združevati ključne besede, Boolove operatorje in napredne operatorje iskalnih kriterijev v enem povpraševalnem nizu iskalnega kriterija[29].

Nekaj primerov podajamo spodaj:

operator	**rezultat**
(as said) AND (don king)	vsi rezultati, ki vsebujejo besedi "as", "said", "don" in "king"
(uefa euro 2016) OR (olympics rio)	vsi rezultati, ki vsebujejo besede "uefa", "euro" in "2016" ali besedi "olympics" in "rio"
(olympics from:cnn) AND (prevent OR postpone)	vsi rezultati, ki jih odda račun @cnn, ki vsebujejo besedo "olympics" in besedo "prevent" ali "postpone"
(elonmusk OR (elon musk)) AND geocode:"40.7127,-74.0059,50km"	vsi rezultati, ki omenjajo račun @elonmusk ali besedi "elon" in "musk" v radiju 50 kilometrov od centra mesta New York City
(peter prevc) OR (petra prevca) OR (petru prevcu) OR (petrom prevcem) OR (petrom prevcom) OR (pero prevc) OR (perota prevca) OR peterprevc	vsi rezultati, ki omenjajo Petra Prevca[30], skakalca na smučeh – z alternativnimi oblikami zapisovanja njegovega imena

Tabela 4.2: Asociativni operator in združevanje

4.1.2 Filtri iskalnih kriterijev

Filtri iskalnega kriterija so namenjeni podrobnejšemu filtriranju rezultatov, ki jih pridobi implementacija modela za pridobivanje podatkov. Model podpira *vključujoče* (inkluzivne) filtre in *izključujoče* (ekskluzivne) filtre. Razlika med vrstama filtrov je, da prvi rezultat spustijo naprej v primeru, če ustreza

[29] Primer je spremljanje pokrivana olimpijskih iger 2016 s strani določenih medijskih računov, recimo CNN, BBC in NBC – "olympics (from:cnn OR from:nbc OR from:bbcnews)".
[30] Ime skakalca je namenoma sklanjano napačno, saj se takó zapisano velikokrat tudi pojavlja v objavah na družbenih omrežjih.

vsaj eni omejitvi filtra, drugi pa samo, če se rezultat *ne ujema z vsemi omejitvami filtra*.

Implementacija modela vsebuje naslednje vrste filtrov:

filter	**vrsta**	**rezultat**
jezikovni filter	vključujoč	vsi rezultati, za katere jezikovni detektor ugotovi ujemanje v jeziku
uporabniški filter	izključujoč	vsi rezultati, ki jih napišejo v filter vpisani uporabniki ali računi
besedni filter	izključujoč	vsi rezultati, ki vsebujejo eno od besed vpisanih v filter
črkovni filter	izključujoč	vsi rezultati, ki vsebujejo eno od Unicode črk vpisanih v filter
pametni filter	izključujoč	vsi rezultati, za katere pametni filter ugotovi, da niso primerni za shranjevanje

Tabela 4.3: Vrste filtrov iskalnih kriterijev

Pametni filtri ne delujejo s primerjanjem fiksnih omejitvenih vrednosti, pač pa vsebino primerjajo dinamično. Tipičen primer pametnega filtra je *število sledilcev v času pisanja*, ki se primerja z minimalno spodnjo mejo, ki jo določi uporabnik sistema. Tak filter rezultat prepusti samo v primeru, če ima uporabnik več od minimalnega števila sledilcev[31].

Točnejšo implementacijo filtrov podaja poglavje 5.

4.1.3 Obveščanje in urniki iskalnih kriterijev

Obveščanje je namenjeno spremljanju izrednih dogodkov, ki niso napovedljivi. Uporabniki lahko nastavljajo obveščanje po različnih kriterijih. V primeru prekoračitve praga kriterija proženja, sistem uporabnika samodejno obvesti o preseženem pragu ter mu pošlje elektronsko sporočilo z vizualizacijo ključnih podatkov.

[31] Ideja filtra je, da uporabnika ne želimo poslušati s sistemom za analizo širokega mnenja javnosti, če ga ne posluša dovolj drugih uporabnikov družbenih omrežij.

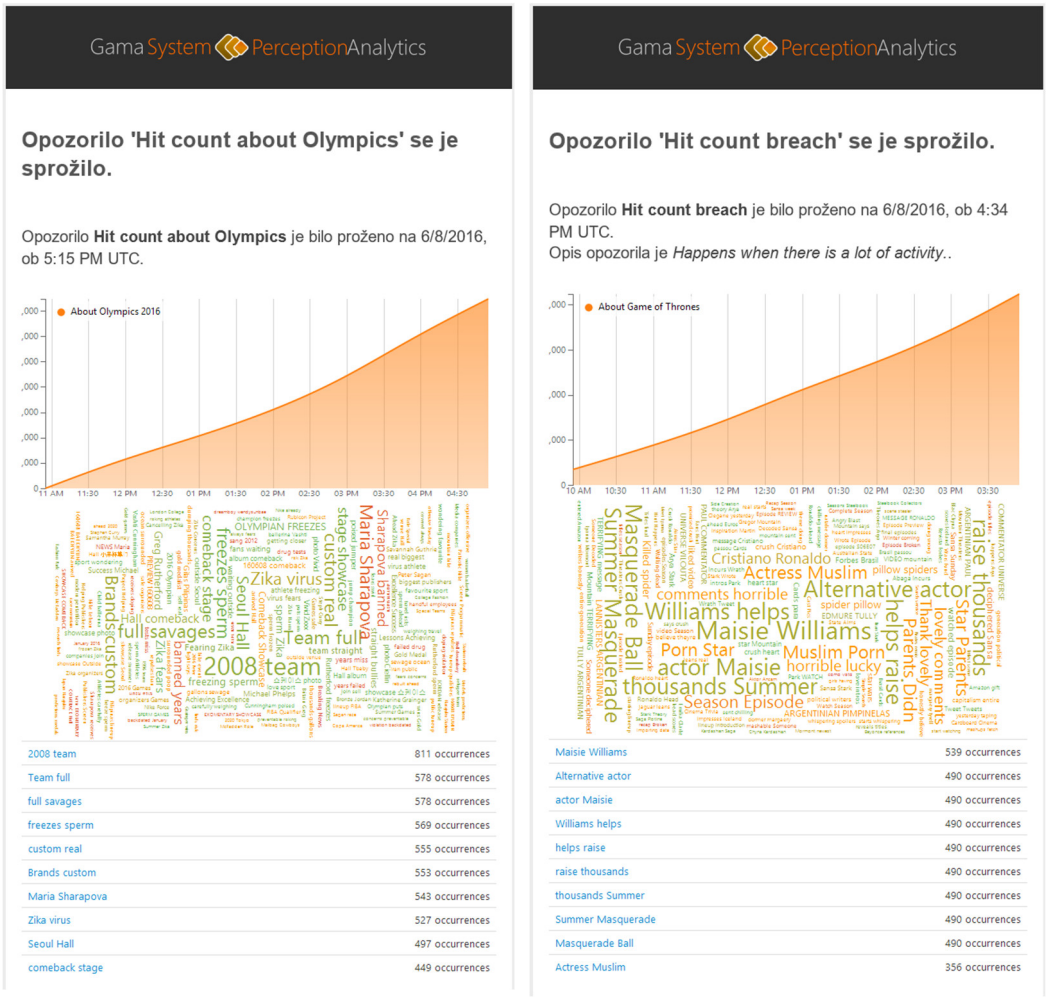
Obveščanje zagotavlja, da uporabniki ne zamudijo ključnih dogodkov znotraj kriterijev, ki jih spremljajo. Trenutna implementacija modela ne omogoča samodejne detekcije hitrih skokov / padcev med analitičnimi podatki.

Kriteriji obveščanja so:

kriterij	**proženje**
število zadetkov	obvestilo se sproži, če je odstopanje v številu zadetkov (število pojavitev) večje od nastavljenega
doseg	obvestilo se sproži, če je odstopanje v dosegu računov ali uporabnikov, ki pošiljajo primerne rezultate za kriterij, večje od nastavljenega
pozitiven sentiment	obvestilo se sproži, če je odstopanje pozitivnega sentimenta zadetkov večje od nastavljenega
negativen sentiment	obvestilo se sproži, če je odstopanje negativnega sentimenta zadetkov večje od nastavljenega
relativen sentiment	obvestilo se sproži, če je odstopanje relativnega sentimenta zadetkov večje od nastavljenega
unikatni uporabniki	obvestilo se sproži, če je odstopanje v številu unikatnih uporabnikov večje od nastavljenega
unikatne lokacije	obvestilo se sproži, če je odstopanje v številu unikatnih lokacij večje od nastavljenega

Tabela 4.4: Kriteriji obveščanja

Primeri samodejnih obvestil, ki jih uporabnik prejme v poštni predal so podani v naslednjih slikah:



Slika 4.3: Primeri samodejnih obvestil

Odstopanje je mogoče v nastavitvah obveščanja omejiti navzgor ali navzdol in kvantificirati z odstotkom drsečih časovnih povprečij.

Tako je mogoče nastaviti obveščanje, ki se sproži na primer ob naslednjih pogojih:

- Število zadetkov je več od 200% višje od povprečja zadnjega tedna
- Negativni sentiment je več od 150% večji od povprečja zadnjega dneva
- Število unikatnih uporabnikov je manj od 50% povprečja zadnjega tedna

Nastavitev takega obveščanja omogoča lovljenje dinamičnih primerov, kjer se o neki témi manjšina uporabnikov zelo frekventno opredeljuje z negativno konotacijo.

Implementacija modela za pridobivanje podatkov omogoča tudi možnost definicije prioritet iskalnih kriterijev, ki povprašujejo izvore po obstoju novih podatkov. Sistem znotraj posamezne prioritete iskanje razporeja po *padajočem času zadnjega iskanja* in tako zagotovi, da se kriteriji, ki so bili posodobljeni najkasneje, najprej osvežijo.

V sistemu obstaja pet (5) definiranih prioritet.

prioriteta	**rezultat**
najnižja	iskanje novih podatkov se izvaja z najnižjo prioriteto
nizka	iskanje novih podatkov se izvaja z nizko prioriteto
normalna	iskanje novih podatkov se izvaja z normalno prioriteto
visoka	iskanje novih podatkov se izvaja z visoko prioriteto
najvišja	iskanje novih podatkov se izvaja z najvišjo prioriteto

Tabela 4.5: Prioritete iskalnih kriterijev

Višja kot je prioriteta, večkrat se sproži izvajanje povpraševanja, kar omogoča ponujanje poslovnih modelov, ki temeljijo na nadstandardnem spremljanju specifične téme v trenutkih, ko se dogodki odvijajo. Primer dviga prioritete za iskalni kriterij je spremljanje širokega odziva javnosti med intervjujem ali pomembnim športnim dogodkom, ki generira velik odziv na družbenih omrežjih. Za vsak urnik zato lahko definiramo začetno prioriteto, končno prioriteto, začetni čas in čas trajanja.

> *Definicija: Začetna prioriteta urnika*
>
> *Začetna prioriteta urnika je prioriteta, ki se nastavi iskalnemu kriteriju ob začetku urnika. Začetna prioriteta vztraja celoten čas urnika.*
>
> *Definicija: Končna prioriteta urnika*
>
> *Končna prioriteta urnika je prioriteta, ki se nastavi iskalnemu kriteriju po poteku časa urnika.*

V ta namen implementacija modela ponuja *urnike prioritet*, ki omogočajo samodejno zvišanje ali znižanje prioritete iskanja ob določenem času in nastavitev časa trajanja. Sistem ob določenem času zviša / zniža prioriteto na začetno prioriteto urnika, počaka določeno časovno obdobje in po koncu obdobja prioriteto nastavi na končno prioriteto urnika. S to funkcionalnostjo uporabniki lahko vnaprej določajo časovne periode, ko je potrebno sistem bolj obremeniti z namenom pridobivanja več rezultatov.

4.1.4 Sistemske nastavitve iskalnih kriterijev

Med sistemske nastavitve iskalnih kriterijev štejemo predvsem nastavitve, ki niso vidne vsem vlogam uporabnikov in so administrativne narave.

Te so:

- Prioriteta iskanja

  Določa trenutno nastavljeno prioriteto iskanja, ki vpliva na razvrščanje kriterijev v fazi določanja naslednjega kriterija za katerega je potrebno izvesti iskanje. Prioritete ročno določajo uporabniki z administratorsko vlogo ali jih samodejno spreminjajo nastavljeni urniki prioritet.

- Čas hrambe rezultatov

  Definira premikajoče časovno okno v katerem hranimo vse najdene rezultate kriterija. Nastavitev je namenjena podpori poslovnemu modelu, ki omogoča različne čase hrambe podatkov in analiz. Poslovni model temelji tako na številu, kot velikosti iskalnih kriterijev.

- Čas hrambe analitičnih podatkov

  Definira premikajoče časovno okno v katerem hranimo vse izračunane analitične rezultate in statistike.

- Omejitev zadetkov v eni iteraciji

  Določa zgornjo mejo števila zadetkov v enem povpraševanju za primer povpraševalnega toka podatkov. Nastavitev je implementacijska podrobnost stopničastega algoritma za omejevanje širine toka kriterija, ki je podrobneje opisan v poglavju 4.2.

- Omejitev števila zaporednih prevelikih iteracij

  Določa število stopnic v stopničastem algoritmu za omejevanje toka kriterija.

- Račun iskalnega kriterija

  Predstavlja administrativno nastavitev za nastavljanje računa kateremu kriterij pripada.

4.1.5 Ostale nastavitve iskalnih kriterijev

Med ostale nastavitve iskalnih kriterijev štejemo predvsem nastavitve, ki niso primarnega pomena.

Te so:

- Prednostni jezik

  Nastavitev kontrolira nagnjenost (bias) detektorja jezikov v primerih analiziranja jezikov, ki so si med seboj zelo podobni. Tipični primeri so španščina (Španija), španščina (Argentina), španščina (Mehika) ali hrvaščina (Hrvaška), bosanščina (Bosna in Hercegovina), srbščina (Srbija).

- Domenski model sentimenta

  Nastavitev omogoča vklop dvostopenjske analize sentimenta z uporabo specifičnega domenskega modela za problemsko domeno ali témo. Domenski model sentimenta se trenira na zajeti vsebini, ki specifično opisuje omenjeno témo in dovoljuje večje zanesljivosti pri preučevanju.

- Omogočen kriterij

  Nastavitev omogoča začasen izklop delovanja iskalnega kriterija.

- Promocijski kriterij

  Nastavitev določa, da je iskalni kriterij promocijske narave in je viden vsem računom v sistemu ter je namenjen promocijski predstavitvi delovanja sistema.

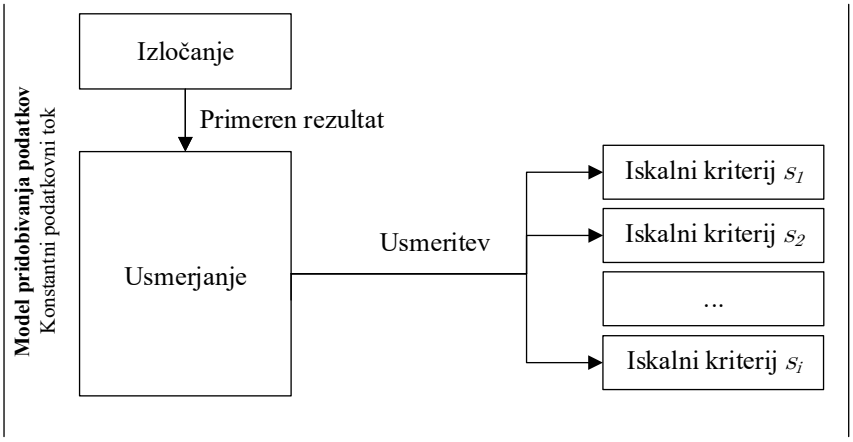
4.2 Proces pridobivanja podatkov

Proces pridobivanja podatkov je z implementacijskega gledišča različen za *konstantni podatkovni tok* in *povpraševalni podatkovni tok*. Prvi dosega bistveno manjšo latenco (L_K, Model pridobivanja podatkov, poglavje 2.2, enačba (2.4)) od drugega, vendar ima tudi bolj okrnjene funkcionalnosti.

Konstantni podatkovni tok *ne izvaja aktivnega povpraševanja*, ampak na podatke čaka in jih pridobiva iz usmerjevalnika založnik-naročnik (pub-sub), katerega opis podajamo v poglavju 5.2. Povpraševalni niz je v tem primeru

pomembno okrnjen, saj ne omogoča naprednih operatorjev[32], pač pa samo uporabo ključnih besed.

Proces pridobivanja podatkov primeru konstantnega povpraševalnega toka je torej naslednji:



Slika 4.4: Proces pridobivanja podatkov – konstantni tok

Ker v primeru konstantnega toka ne moremo govoriti o aktivnem pridobivanju podatkov, pač pa o *pasivnem poslušanju*, je nesmiselno govoriti o izvedbi procesa pridobivanja podatkov, saj je mehanizem usmerjanja hkrati tudi mehanizem pridobivanja.

Nasprotno za zagotavljanje povpraševalnega podatkovnega toka uporabljamo *aktivno povpraševanje* po novih podatkih neposredno z uporabo aplikacijskega vmesnika ponudnikov.

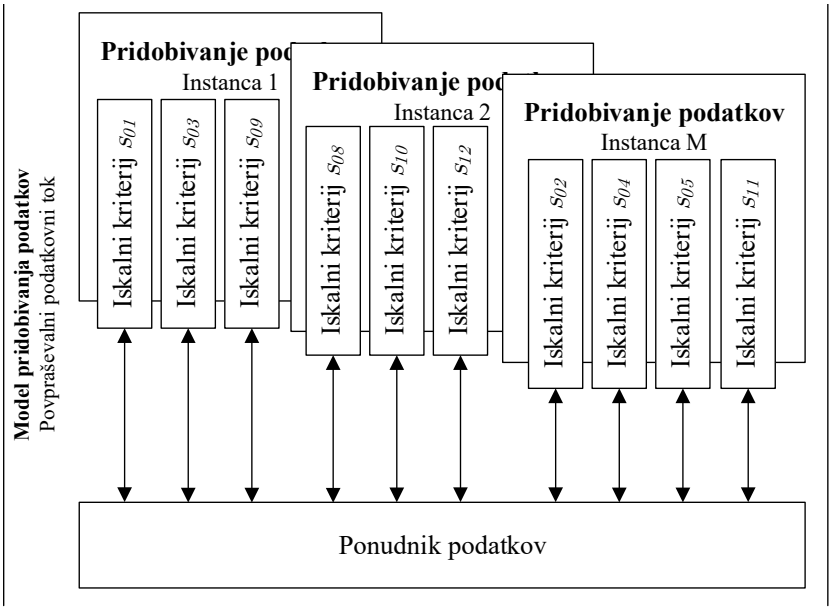
Koncept povpraševanja temelji na večjem številu instanc črpalk (pump), ki vzporedno, z več lokacij in različnimi izvornimi naslovi IP črpajo podatke od ponudnika storitve.

> *Definicija: Črpalka*
>
> *Črpalka je proces, ki teče neprekinjeno v zanki in prenese nove podatke za N iskalnih kriterijev hkrati, kjer je N nastavljiva vrednost, ki definira stopnjo paralelnosti črpalke.*

Izvedba procesa pridobivanja podatkov za povpraševalni podatkovni tok je torej izvedena z več instancami (recimo M instancami), kjer vsaka instanca

[32] Operatorji kot so *from*, *to*, *since* in asociativni operator v konstantnih tokovih niso podprti zaradi omejitev vmesnika ponudnikov storitev.

vsebuje črpalko, ki prenaša podatke za N iskalnih kriterijev hkrati. Spodnja slika prikazuje tipičen proces pridobivanja podatkov:



Slika 4.5: Proces pridobivanja podatkov – povpraševalni tok

Zgornja slika prikazuje stopnjo paralelnosti štiri (4), kjer vsaka instanca vzporedno pridobiva podatke za štiri iskalne kriterije. Stopnja paralelnosti se nastavlja glede na strojne sposobnosti instanc, ki so za dotično postavitev sistema zakupljene v oblaku.

Psevdo algoritem za pridobivanje podatkov je naslednji:

```
01   while true do
02      A ← select accessor
03      S ← get N search criteria
04      T ← max results per query
05      I ← max results per iteration of query
06      R ← {} // current results
07      lock A
08      for all s in S do in parallel
09            lock s
10            ID ← last stored result identifier for s
11            while |R| < T do
12                  N ← get I new results from source
13                  R ← R ∪ N
14            end while
15            last found result identifier for s → ID
16            unlock s
```

```
17              store results for s
18              check and update throttling for s
19          end for
20          unlock A
21      end while
```

Algoritem 4.1: Pridobivanje podatkov

Algoritem prikazuje implementacijo pridobivanja podatkov za povpraševalni tok. Vrstice 02 – 06 definirajo predpogoje za izvedbo glavne zanke, ki pridobi vse podatke povpraševanja (vrstice 08 – 17). Pred vstopom v glavno zanko, v vrstici 07, se dostopnik zaklene (zaklepanje in vzroki zanj so opisani v poglavju 4.2.2). V vrstici 09 se zaklene iskalni kriterij. Poizvedbe se izvajajo v več iteracijah (vrstice 11 – 13). Iskalni kriterij se odklene v vrstici 15. V zadnji, 18. vrstici, se odklene tudi dostopnik.

Iz zgornjega algoritma je razvidno tudi, da se povpraševanja v vsaki instanci izvajajo ves čas (vrstica 01) – instanca po končanju enega povpraševanja takoj prične z novim ciklom.

4.2.1 Pridobivanje podatkov in omejitve ponudnikov

Vsa večja družbena omrežja preko svojega aplikacijskega vmesnika dovoljujejo dostop do večine javnih podatkov svojih uporabnikov, njihovih povezav z drugimi uporabniki, komunikacij, lastnosti profila, itd. Aplikacijski vmesniki so relativno dobro dokumentirani[33], hkrati pa točno določajo pravila uporabe, ki so namenjena omejevanju količine prometa, ki ga dovolijo zunanjim aplikacijam.

Večina vmesnikov omejuje število dovoljenih povpraševanj v časovnem obdobju in to preverja preko avtenticiranega računa, ki je uporabljen za dostop do vmesnika. Hkrati ponudniki spremljajo promet glede na izvorne naslove IP in račun trajno ali začasno onemogočijo v primerih, ko je količina povpraševanj prevelika.

V ta namen smo v modelu pridobivanja podatkov implementirali porazdeljeno povpraševanje, kjer navidezni odjemalci uporabljajo vmesnike API iz različnih lokacij na internetu (različnih podatkovnih centrov v oblaku) in z

[33] https://developers.facebook.com/docs, https://dev.twitter.com/rest/public, https://dev.twitter.com/streaming/overview

različnimi računi, ki se uporabljajo za avtentikacijo. Te račune in implementacijo povpraševanja z njimi imenujemo *dostopniki* (accessors). Uporabljamo jih pri poizvedovanju za nove podatke iskalnih kriterijev, ki se izvajajo vzporedno in skrbno spremljamo, da dostopniki ne kršijo omejitev uporabljenega vmesnika ponudnika.

Dostopniki imajo naslednje lastnosti:

- Avtentikacijske podatke računa

  Avtentikacijski podatki so klasični podatki za prijavo preko vmesnika in so različni od ponudnika do ponudnika

- Skupno število povpraševanj

  Ponudniki, ki omejujejo število povpraševanj v določenem času, račun ob prekoračitvi neposredno blokirajo. Dostopniki imajo zato shranjeno maksimalno število povpraševanj, ki so jim dovoljena.

- Število povpraševanj na voljo

  Število povpraševanj, ki so trenutno še na voljo – pred iztekom časa in ponastavitvijo dostopnika s strani ponudnika.

- Čas ponastavitve

  Čas naslednje ponastavitve podatkov dostopnika. Ob ponastavitvi se dostopnikovo število povpraševanj na voljo izenači s skupnim številom povpraševanj in dostopnik se *napolni*.

- Trenutna aktivnost

  Zastavica ali je zastopnik trenutno zaposlen z iskanjem v kateri koli instanci.

- Čas zadnjega začetka iskanja

  Čas zadnjega začetka iskanja oziroma aktivacije dostopnika.

- Čas zadnjega konca iskanja

  Čas zadnjega konca oziroma deaktivacije dostopnika.

- Dostopnik omogočen

  Zastavica ali je zastopnik omogočen ali onemogočen. V primeru, če ponudnik storitve blokira dostopnik in vrne ustrezen odgovor, ga

sistem onemogoči v izogib ponovni uporabi in še hujšim povračilnim ukrepom s strani vmesnika družbenega omrežja.

4.2.2 Izbira dostopnika

Proces pridobivanja podatkov (algoritem (4.1)) v vrstici 01 izbira dostopnik, ki bo uporabljen za pridobivanje podatkov v trenutni izvedbi iskanja novih rezultatov za iskalni kriterij.

Definicija: Zaklenjen dostopnik

Dostopnik je zaklenjen, kadar je v uporabi v drugi instanci, ki ga uporablja za svoje trenutno izvajajoče povpraševanje po novih podatkih.

Dostopnik se izbere po več kriterijih:

- Za izbor se upoštevajo samo *omogočeni dostopniki*

  Algoritem za izbiro dostopnika ne izbira med onemogočenimi dostopniki v izogib uporabi neveljavnih avtentikacijskih podatkov ali s strani ponudnika deaktiviranega računa.

- Za izbor se upoštevajo samo *odklenjeni dostopniki*

  Trenutno že aktivni dostopniki se ne štejejo med primerne kandidate, saj opravljajo delo v eni od preostalih instanc. Algoritem za izbiro jih zato izključi.

- Izbere se najboljši kandidat glede na *število povpraševanj na voljo*

  Algoritem za izbiro uredi dostopnike po padajočem številu povpraševanj, ki jih imajo na voljo in izbere prvega. Razlog za to je, da se vedno uporabijo *najbolj polni* dostopniki, saj imajo največjo verjetnost[34] dokončanja dela.

Algoritem za izbiro dostopnika je tako naslednji:

```
01    sa ← {}  // selected accessor
02    SA ← get all accessors
03    M ← 0    // current maximum remaining accesses
04    for all s in SA do
```

[34] Število iteracij v katerih bo dostopnik uporabljen ni znano vnaprej, saj je nemogoče predvideti količine novih rezultatov, ki jih bo dostopnik našel. Vsled temu se vedno izberejo dostopnik z največjim številom preostalih povpraševanj.

```
05      if s is not enabled then
06            if s is not locked then
07                  if remaining accesses of s > M then
08                        M ← remaining accesses of s
09                        sa ← s
10                  end if
11            end if
12      end if
13    end for
14    lock sa
15    return sa
```

Algoritem 4.2: Izbira dostopnika

Algoritem za izbiro dostopnika se sprehodi čez vse dostopnike (vrstice 04 – 13) in pri vsakem v vrstici 05 preveri ali je omogočen. V vrstici 06 preveri ali je zaklenjen – trenutno v uporabi v drugi instanci. Algoritem potem poišče dostopnik, ki ima maksimalno število povpraševanj na voljo in ga zaklene za uporabo (vrstica 14).

4.2.3 Izbira in zaklepanje iskalnih kriterijev

Po izbiri dostopnika je potrebno izbrati iskalne kriterije za katere bo instanca iskala nove rezultate v trenutni izvedbi povpraševanja. V krovnem algoritmu povpraševanja (algoritem 4.1) se izbira kriterijev izvede v vrstici 02.

Glede na to, da več instanc hkrati opravlja povpraševanja (slika 4.5), mora trenutna instanca izbrati kriterije, za katere se trenutno ne izvaja nobeno povpraševanje s strani katere koli druge instance. – iskalni kriterij mora biti odklenjen. Analogno dostopnikom se tudi iskalni kriteriji zaklepajo[35] v času, ko se zanje pridobivajo novi rezultati.

Obratna pot, torej dovoljevanje izvajanja povpraševanj za isti iskalni kriterij s strani več instanc, bi pripeljalo do pridobivanja duplikatov rezultatov.

[35] Zaklep dostopnika ali iskalnega kriterija *nima vpliva* na uporabniško izkušnjo. Namenjen je izključno zagotavljanju pravilnost delovanja vzporednih povpraševanj z več instancami. Z iskalnimi kriteriji je, tudi kadar so zaklenjeni, mogoče upravljati nemoteno.

Definicija: Zaklenjen iskalni kriterij

Iskalni kriterij je zaklenjen ves čas, ko se izvaja pridobivanje novih rezultatov. Namen zaklepanja kriterija je zagotavljanje, da povpraševanje za posamezni iskalni kriterij izvaja samo ena instanca.

Izbira iskalnih kriterijev uporablja drugačen algoritem od izbire dostopnika, saj je glede na definicijo modela potrebno določiti iskalne kriterije, ki so bili znotraj prioritete kateri pripadajo (poglavje 4.1.4), posodobljeni zadnji.

Algoritem za izbiro iskalnih kriterijev je naslednji:

```
01    sc ← {}  // selected search criteria set
02    D ← degree of parallelism
03    SC ← get all search criteria
04    for i = 1 to |SC| do
05       if SC[i] is disabled or locked then
06             remove SC[i] from SC
07       end if
08    end for
09    sort SC by last search time, ascending
10    rank based on search criteria priorities
11    for i = 1 to D do
12       sc ← sc ∪ SC[i]
13       lock sc[i]
14    end for
15    return sc
```

Algoritem 4.3: Izbira iskalnih kriterijev

Algoritem izbere D iskalnih kriterijev, kjer je D stopnja paralelizma, ki jo uporabljajo instance za sočasno (paralelno) povpraševanje znotraj posamezne instance. Algoritem v vrsticah 02 – 03 pridobi stopnjo paralelizma in seznam vseh kriterijev. V vrsticah 04 – 08 najprej odstrani vse onemogočene in zaklenjene kriterije. V vrstici 09 izvede sortiranje po zadnjem času iskanja in pridobi kriterije, ki so bili posodobljeni najdlje nazaj. V vrstici 10 izvede rangiranje glede na prioritete iskalnih kriterijev in v ospredje potisne kriterije, ki imajo višjo prioriteto. Končna izbira se izvede v vrsticah 11 – 14, kjer iz obstoječega nabora izbere D kriterijev.

4.2.4 Paralelnost povpraševanj

Izvedbo paralelnosti povpraševanj smo implementirali na dveh nivojih, kot to prikazuje slika 4.5. Nabor iskalnih kriterijev v sistemu porazdeljujemo *preko vseh instanc* in *znotraj posamezne instance*.

Namen porazdeljevanja je doseganje višje *iskalne prepustnosti* in zmanjševanje intervala zakasnitve (enačba 3.19).

Implementacija znotraj posamezne instance je trivialna in je izvedena s paralelno zanko. Implementacija preko meje instanc pa zahteva uvedbo zaklepanja iskalnih kriterijev, kar je opisano v prejšnjem poglavju. Sistemske statistike, ki so prisotne v administrativnem delu rešitve omogočajo vpogled v trenutno stanje porazdeljevanja iskalnih kriterijev:

Statistika sistema Trenutna statistika sistema

Interval osvežitve rezultatov (najvišja prioriteta)	/	Interval osvežitve za najvišjo prioriteto iskanja.
Interval osvežitve rezultatov (najvišja prioriteta)	/	Interval osvežitve za višjo prioriteto iskanja.
Interval osvežitve rezultatov (normalna prioriteta)	13 sekund	Interval osvežitve za normalno prioriteto iskanja.
Interval osvežitve rezultatov (nižja prioriteta)	/	Interval osvežitve za nižjo prioriteto iskanja.
Interval osvežitve rezultatov (najnižja prioriteta)	/	Interval osvežitve za najnižjo prioriteto iskanja.
Število iskalnih kriterijev v teku	30	Trenutno število iskalnih kriterijev v teku.
Število iskanj v teku	24	Trenutno število iskalnih kriterijev v fazi iskanja.
Število shranjevanj v teku	6	Trenutno število iskalnih kriterijev v fazi shranjevanja.
Število iskalnih ključev v teku	3	Trenutno število iskalnih ključev v uporabi.
Število rezultatov	9.368.659	Trenutno število shranjenih rezultatov.
Število uporabnikov	27	Trenutno število uporabnikov.
Število iskalnih kriterijev	141	Trenutno število iskalnih kriterijev za vse uporabnike.
Število omogočenih iskalnih kriterijev	119	Trenutno število omogočenih iskalnih kriterijev.
Število onemogočenih iskalnih kriterijev	22	Trenutno število onemogočenih iskalnih kriterijev.
Število promocijskih iskalnih kriterijev	9	Trenutno število promocijskih iskalnih kriterijev.
Število izbrisanih iskalnih kriterijev	0	Trenutno število iskalnih kriterijev, ki čakajo na uničenje.
Število analiz zadetkov	6.159.786	Trenutno število sistemsko generiranih in uporabniško definiranih analiz zadetkov.
Število analiz dosega	6.110.998	Trenutno število sistemsko generiranih in uporabniško definiranih analiz dosega.
Število analiz sentimenta	6.092.653	Trenutno število sistemsko generiranih in uporabniško definiranih analiz sentimenta.
Število preostalih iskalnih ključev	63%	Trenutni procent nezasedenih iskalnih ključev.

Slika 4.6: Sistemske statistike porazdeljevanja iskalnih kriterijev[36]

[36] *Število iskalnih kriterijev v teku* predstavlja število zaklenjenih iskalnih kriterijev, od katerih *število iskanj v teku* dejansko išče, *število shranjevanj v teku* pa nove rezultate trenutno shranjuje. *Število iskalnih ključev* predstavlja število zaklenjenih dostopnikov.

4.2.5 Iteriranje

Praktično vsi ponudniki podatkov omejujejo maksimalno število pridobljenih rezultatov z enim klicem vmesnika API. Navadno je omejitev od 100 – 1000 rezultatov, ki jih je mogoče pridobiti z izvedbo enega klica. Ponudniki omogočajo tudi ostranjevanje (paging) rezultatov ter na ta način podpirajo pridobivanje več od omenjenega števila rezultatov z zaporednimi klici na vmesnik API.

V našem primeru smo v izogib tej omejitvi implementirali iterativno, zaporedno izvajanje povpraševanj, kar omogoča pridobivanje večjega števila rezultatov znotraj enega (logičnega) povpraševanja.

Algoritem iterativnega povpraševanja je naslednji:

```
01    T ← max results per query
02    I ← max results per iteration of query
03    s ← current search criterium
04    p ← fill factor [0, 1]
05    R ← {} // current results
06    ID ← last stored result identifier for s
07    N ← get I new results from source
08    while |I| ≥ (p × I) and |R| < T do
09       R ← R ∪ N
10       N ← get I new results from source
11       if |I| < (p × I) then
12             R ← R ∪ N
13             exit while
14       end if
15    end while
16    last found result identifier for s → ID
```

Algoritem 4.4: Iterativno iskanje

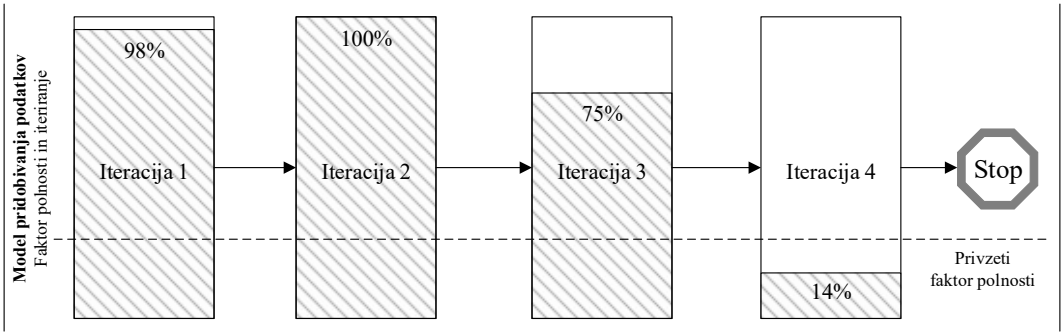
Algoritem iteriranja v vrsticah 01 – 05 pridobi podatke za izvedbo glavne zanke. Posebnost je vrstica 04, ki pridobi *faktor polnosti* (fill factor) oziroma odstotek rezultatov v posamezni iteraciji, ki mora biti presežen za uspešno nadaljevanje naslednje iteracije. Iskanje se začne od zadnjega že najdenega rezultata naprej (vrstici 06, 07).

Glavna zanka išče nove rezultate dokler ni doseženo maksimalno število najdenih rezultatov ali pa je najdeno premalo rezultatov v iteraciji, kar pomeni,

da večjega števila novih rezultatov ni več – vrstice 08 – 14. Specifično se faktor polnosti primerja znotraj vsake iteracije s privzetim faktorjem polnosti[37] p. V primeru manjšega števila rezultatov se zanka prekine (vrstice 11 – 13).

Definicija: Faktor polnosti

Faktor polnosti je odstotek maksimalnega števila rezultatov iteracije, ki je populiran z rezultati in nakazuje možen obstoj dodatnih rezultatov. Nova iteracija se izvede samo in le v primeru, ko je faktor polnosti trenutne iteracije večji od privzetega.



Slika 4.7: Faktor polnosti in iteriranje

Algoritem se zaključi v vrstici 16, kjer shranimo identifikator zadnjega najdenega rezultata, ki bo v pomoč naslednjemu povpraševanju oziroma instanci, ki ga bo izvedla.

4.3 Omejevanje toka

Zaradi količine podatkov, ki jih je mogoče zahtevati s preprostimi iskalnimi kriteriji, smo implementirali algoritem za omejevanje toka (flow throttling), ki je vezan na posamezni kriterij. Tipični uporabniški scenarij ob spoznavanju storitve je vpis povpraševalnega niza, ki generira zelo širok tok podatkov[38]. V primeru neprekinitve takega toka je poraba virov previsoka, da bi upravičila uporabnikov namen.

[37] Privzeti faktor polnosti je 0,25, kar se je empirično izkazalo za dobro mejno vrednost prekinitve iteriranja.

[38] Prvi iskalni niz je *bmw*, *apple* ali *microsoft*. Taki kriteriji generirajo preko 100.000 rezultatov vsako uro.

Ustaljeno omejevanje tokov [24] omogoča definicijo prepustnosti na način, da podatke v toku *upočasnjuje* ali *vzorči* in s tem pogojno prepušča. S tem kontrolira širino toka na sledeči način:

(1) Upočasnjevanje

Preširok tok pomeni upočasnitev pretoka podatkov v toku. Tak način se uporablja predvsem v primerih omejevanja *končnega* toka podatkov. V našem primeru je tok podatkov neskončen in upočasnjevanje niti ni zaželeno, niti ni primerno.

(2) Vzorčenje

Preširok tok pomeni izpuščanje podatkov po nekem vzorcu. Navadno je vzorčenje omejeno na odstotek prepuščenih podatkovnih paketov. V naprednejših omejevalnih mehanizmih se omogočajo tudi scenariji z možnostjo analize vsebine in zvišanjem verjetnosti prepuščanja pomembnejših podatkov. Vzorčenje omogoča kontrolo širine toka, vendar ne zagotavlja pridobivanja vseh podatkov, ki so bili originalno prisotni v toku. Tudi vzorčenje, kot mehanizem omejevanja toka za našo problemsko domeno ni primerno, saj načrtno izgublja rezultate.

Omejevanje je zato implementirano s *večstopenjskih, koračnim preverjanjem širine toka*. Pri ugotavljanju ali je tok preširok zato temeljimo na dveh ključnih podatkih: *številu rezultatov pri posameznem povpraševanju* in *številu prekoračitev*. Ideja implementacije je preverjanje števila pridobljenih rezultatov v minutnih intervalih ter koračno štetje prekoračenih vrednosti. V primeru prekoračitve zadnjega koraka, se kriterij onemogoči.

Psevdo algoritem za omejevanje toka je tako naslednji:

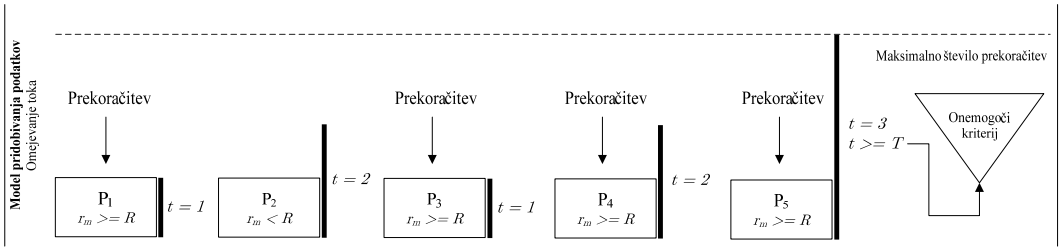
```
01    s ← current search criterium
02    perform search operation
03    r ← number of search results in this operation
04    R ← number of search results allowed per minute
05    d ← duration of this search operation in milliseconds
06    t ← get current step threshold for s
07    T ← get maximum step threshold for s
08    // number of search results in this operation per minute
09    rm ← (r/d) × 1000 × 60
10    if rm ≥ R then
11       increase t for s
12    else
```

```
13        decrease t for s
14     end if
15     store t
16     if t ≥ T then
17        disable s
18     end if
```

Algoritem 4.5: Koračni algoritem za omejevanje toka

Algoritem v vrsticah 01 – 05 opravi povpraševanje in pridobi čas operacije iskanja in število pridobljenih rezultatov. Vrstici 05 in 06 pridobita trenutno stanje omejevanja za iskalni kriterij. V vrstici 09 se izračuna povprečno število pridobljenih rezultatov na minuto za opravljeno iskanje, ki se v vrsticah 10 – 14 primerja z nastavljenim. V primeru, ko pride do prekoračitve se korak poveča, nasprotno pa zmanjša. Po končanju operacije iskanja se preveri ali je prišlo do prekoračitve dovoljenih korakov za nadaljnje spremljanje toka (vrstice 16 – 18) in v tem primeru onemogoči kriterij.

Grafični pogled na algoritem je podan v naslednji sliki:



Slika 4.8: Večstopenjsko koračno omejevanje toka

Slika prikazuje zaporedna povpraševanja (P_1 – P_5) istega iskalnega kriterija, kjer v povpraševanjih P_1, P_3, P_4 in P_5 pride do prekoračitve širine toka, kar posledično zviša število prestopov (t) iskalnega kriterija. Po ustreznem številu prestopov, ki je večje od števila dovoljenih prestopov na tem kriteriju (T), se kriterij onemogoči, uporabnika pa se o tem obvesti preko elektronske pošte.

Opisna implementacija omejevanja podatkovnega toka ne pomeni, da uporabniki ne morejo zahtevati poljubno širokih podatkovnih tokov – nasprotno. Z implementacijo algoritma za omejevanje toka lahko natančno kontroliramo kako širok tok je uporabniku *dovoljen*, ter ga obvestimo, ko je ta tok presežen in kriterij posledično onemogočen.

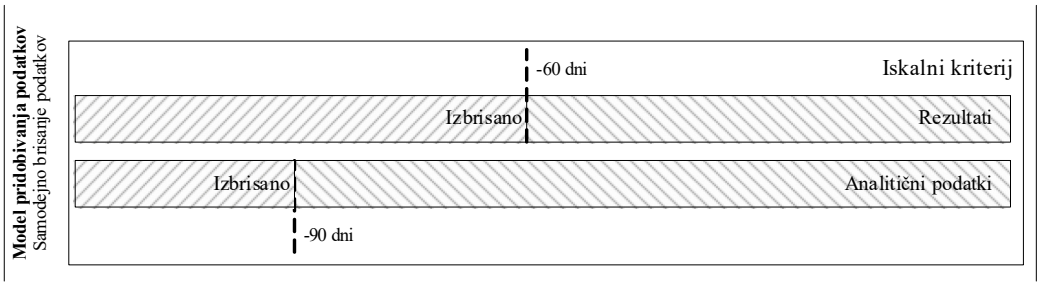
Omejevanje pretoka podatkov je tudi del poslovnega modela storitve – uporabniki, ki želijo širše tokove podatkov, načeloma za storitev plačajo več. Poslovni model storitve opisujemo v poglavju **Error! Reference source not found.**.

4.4 Samodejno brisanje podatkov

Količina podatkov shranjenih za iskalne kriterije, ki imajo dovoljen širok tok podatkov lahko precej naraste. V ta namen smo implementirali samodejno brisanje podatkov z uporabo drsečega časovnega okna oziroma drsečega roka hrambe.

Vsak kriterij ima dve lastnosti, ki vsebinsko pokrivata različen nabor podatkov:

- Čas hrambe rezultatov
- Čas hrambe analitičnih podatkov



Slika 4.9: Samodejno brisanje podatkov

Samodejno brisanje enostavno odreže[39] (truncate) vse rezultate (in shranjene analitične podatke), ki so ob času izvajanja samodejnega brisanja starejši od določenega števila dni. Namen brisanja je obvladljivost shranjene količine rezultatov in analitičnih podatkov za vsak posamezni iskalni kriterij.

4.5 Porazdeljevanje podatkov

Glede na to, da se vsi podatki iskalnih kriterijev, analitični podatki in sistemski podatki shranjeni v podatkovni bazi v oblaku, smo uporabili načine za

[39] Zaradi trivialnosti implementacije algoritma za brisanje podatkov ne podajamo.

skaliranje podatkovnega nivoja z uporabo porazdeljenih podatkovnih baz z deljenjem (sharding) podatkov.

Deljenje podatkov je tehnika za distribucijo velikih količin podatkov, ki imajo podobno strukturo, v več neodvisnih podatkovnih baz. Posebno primerna je za večodjemne modele (multi-tenant models), kjer različne stranke svoje podatke shranjujejo v eni logični podatkovni bazi.

V našem primeru sta vzroka za potrebo po porazdeljevanju podatkovnega nivoja dva:

- Velikost podatkov

  Količina podatkov[40] je prevelika tudi za maksimalne RDBMS podatkovne baze, ki jih ponudniki ponujajo v oblaku[41]. Ker v sistemu hkrati spremljamo več sto kriterijev lahko hitri naletimo na zgornjo mejo.

- Hitrost procesiranja in omejitve zaklepanja

  Porazdeljevanje na več podatkovnih baz seveda ugodno vpliva na hitrost procesiranja, saj se podatkovne baze s tem porazdeljujejo tudi na več strežnikov. Dodatne prednosti, ki bistveno doprinesejo k pohitritvam dosegamo tudi zaradi manjše količine zaklepanja RDBMS entitet, kot so indeksi, tabele in vrstice podatkovne baze, saj se nahajajo v ločenih podatkovnih bazah.

Deljenje na več podatkovnih baz se izvaja s polno ohranitvijo referenčne integritete in združevanjem podobnih podatkov v isti delitvi (shard) podatkovne baze. Za uspešno delitev je najpomembnejša primerna izbira delilnega ključa (sharding key), ki določa v katero podatkovno bazo posamezen podatek spada.

Definicija: Delilni ključ

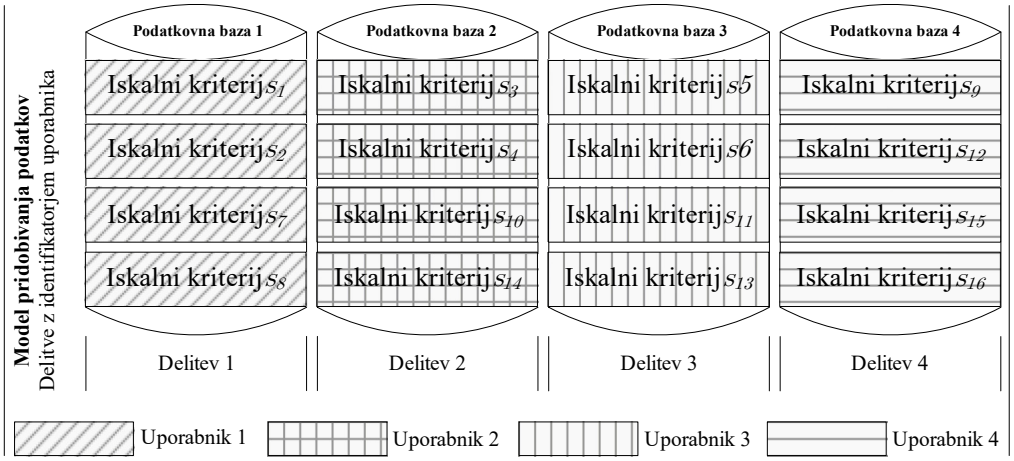
Delilni ključ je polje podatkovne baze, ki v ključni tabeli predstavlja primarni ključ. Posledično so tudi podatki v vseh tujih tabelah, ki ključno

[40] Podatki posameznega iskalnega kriterija lahko zasedajo tudi 50 GB in več.
[41] V času pisanja je največja dovoljena podatkovna baza v Azure Sql Database ponudbi omejena na 1 TB.

tabelo referencirajo s tujimi ključi, shranjeni v isti delitvi podatkovne baze.

Kandidata za delilni ključ v našem primeru sta dva: *identifikator uporabnika* (user identifier) in *identifikator iskalnega kriterija* (search criteria identifier). Problem izbire identifikatorja uporabnika je v tem, da s tem bistveno bolj omejimo količino dovoljenih podatkov za posameznega uporabnika[42] ali podjetje, ki storitev uporablja. Hkrati so zmogljivosti procesiranja za celotno stranko v tem primeru omejene na uporabo ene podatkovne baze.

Spodnja shema prikazuje ta primer:



Slika 4.10: Delitve z identifikatorjem uporabnika

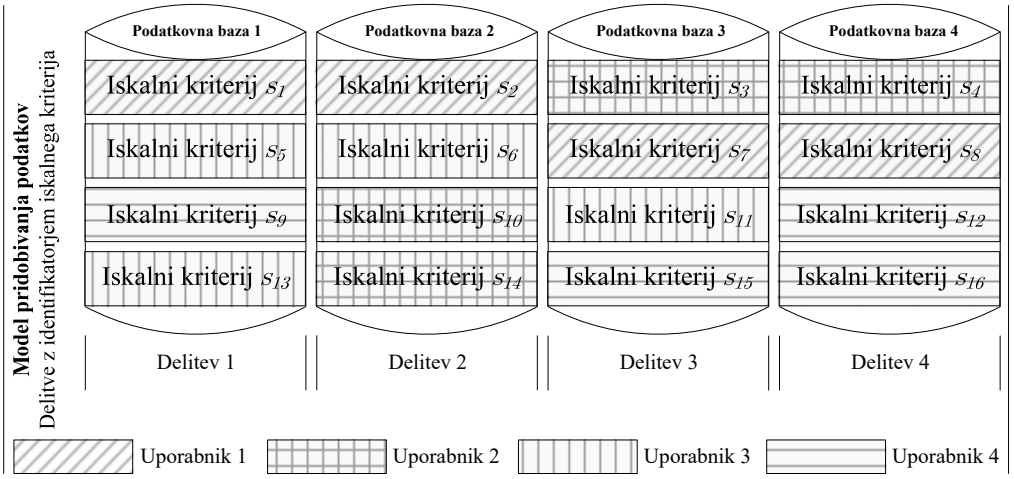
Izbira identifikatorja iskalnega kriterija za delilni ključ je učinkovitejša. Omogoča porazdeljevanje bremena in podatkov posamezne stranke na več podatkovnih baz. To je sicer tudi manjša slabost tega pristopa, saj se vsi podatki stranke ne nahajajo v isti podatkovni bazi, kar v določenih primerih zahtevo uporabo širokih povpraševanj[43] (fan-out queries).

Uporabljen je bil najprimernejši delilni ključ za našo problemsko domeno – *identifikator iskalnega kriterija*. Posledično so vsi podatki, ki se dotikajo istega iskalnega kriterija vedno shranjeni v isti podatkovni bazi, iskalni kriteriji uporabnika pa porazdeljeni čez več podatkovnih baz. Glede na to, da je način generiranja identifikatorja iskalnega kriterija naključen se naključno

[42] Uporabniki ali podjetja, ki storitev uporabljajo imajo lahko veliko iskalnih kriterijev.
[43] Pridobivanje seznama vseh iskalnih kriterijev stranke (uporabnika) je tipičen primer širokega povpraševanja, saj je potrebno vzporedno vprašati vse delitve.

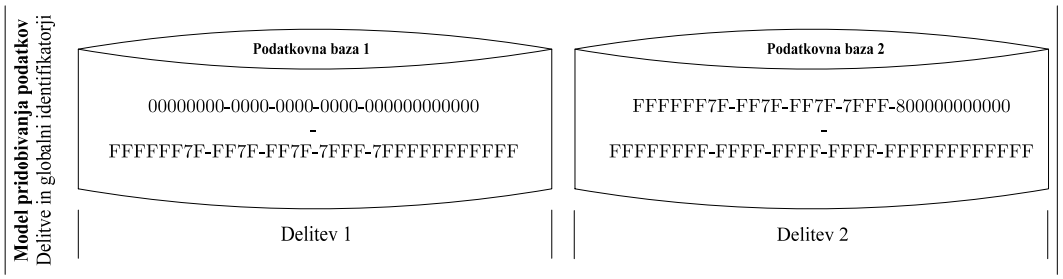
in (enakomerno čez vse podatkovne baze) porazdelijo tudi podatki iskalnega kriterija.

Spodnja shema prikazuje deljenje podatkov med podatkovnimi bazami z uporabo identifikatorja iskalnega kriterija kot delilnega ključa. Različni uporabniki so prikazani črtkano:



Slika 4.11: Delitve z identifikatorjem iskalnega kriterija

Identifikator iskalnega kriterija je polje tipa GUID (globally unique identifier). Način generiranja identifikatorja je standarden, torej nakljućen. Spodnja shema prikazuje primer, ko so uporabljeni dve (2) delitvi:



Slika 4.12: Delitve in globalni identifikatorji

Opisan način porazdeljevanja podatkov ima torej naslednje lastnosti:

- Neomejeno število podatkovnih baz

  Število podatkovnih baz je neomejeno. Implementatorji sistema lahko podatke delijo v poljubno število podatkovnih baz in tako zagotovijo neomejeno rast sistema, ne glede na število uporabnikov. S povečevanjem števila podatkovnih baz se linearno povečujejo tudi performančne sposobnosti podatkovnega nivoja.

- Omejena količina podatkov za en iskalni kriterij

  Količina podatkov, ki jih lahko shrani en iskalni kriterij je trenutno omejena na maksimalno velikost podatkovne baze, kar ne predstavlja resne omejitve.

- Uteženo deljenje kriterijev čez podatkovne baze

  Deljenje kriterijev je uteženo in porazdeljeno čez vse podatkovne baze enakomerno, kar smo dosegli z implementacijo delilnega ključa po identifikatorju iskalnega kriterija.

- Večinski dostop do posamezne podatkovne baze

  Uporabniki za analizo enega iskalnega kriterija dostopajo do podatkov, ki so shranjeni v samo eni podatkovni bazi, njihov seznam kriterijev pa je lahko shranjen v več delitvah.

5 Model izločanja, usmerjanja in filtriranja podatkov

V poglavju podajamo implementacijo modela za izločanje, usmerjanje in filtriranje podatkov, ki je namenjen izločanju neželenih podatkov, usmerjanju podatkov v pravilne iskalne kriterije in filtriranju podatkov, ki v iskalnih kriterijih predstavljajo neželene rezultate.

5.1 Izzivi izločanja in filtriranja v svetu masovnih podatkov

Zaradi ogromne količine vhodnih podatkov obstaja potreba po učinkovitem izločanju in filtriranju. V splošnem pogledu je izločanje proces, kjer veliko večino podatkov enostavno zavržemo, saj niso primerni za nadaljnjo obravnavo – niso povezani s problemsko domeno. Filtriranje, po drugi strani, v našem primeru pomeni dodatno eliminacijo rezultatov, ki so primerni za domeno in so bili usmerjeni na pravi iskalni kriterij, vendar zaradi določenih razlogov ne smejo predstavljati veljavne osnove za analize.

Opisane operacije smo izvedli z implementacijo dveh ogrodij:

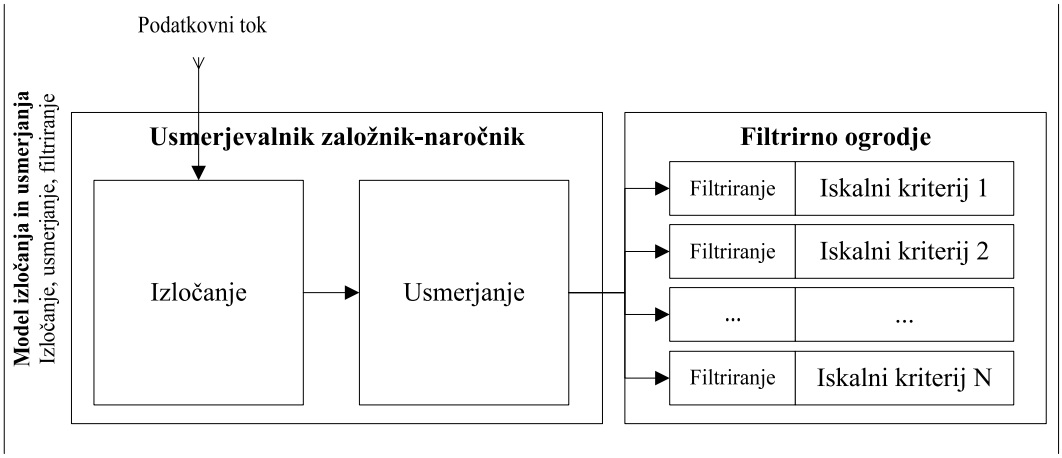
- Usmerjevalnik založnik – naročnik

  Usmerjevalnik opravlja operaciji izločanja in usmerjanja. Izločanje podatkov, na katere ni nihče naročen je osnovna funkcionalnost vseh mehanizmov založnik – naročnik. Usmerjanje je izvedeno z implementacijo naročniških têrminov. Več o usmerjevalniku opisujemo v poglavju 5.2.

- Filtrirno ogrodje

  Filtrirno ogrodje implementira operacijo filtriranja in omogoča dodatno izločanje rezultatov glede na nastavitve posameznega kriterija.

Spodnja shema prikazuje relacije med obema implementiranima ogrodjema:



Slika 5.1: Ogrodja za izločanje, usmerjanje in filtriranje

S stališča podatkovnih tokov (glej sliko 2.2 v poglavju 2.1) se v modelu izločanja, usmerjanja in filtriranja podatkov dotikamo naslednjih podatkovnih tokov:

- Konstantni podatkovni tok

  Na konstantnem podatkovnem toku se izvaja celotno izločanje, saj se v njem nahaja celotna dosegljiva vsebina primernih rezultatov. Velika večina podatkov se izloči, ohranijo se samo podatki, ki ustrezajo vsaj enemu iskalnemu kriteriju. Primerni podatki se usmerijo v ustrezne iskalne kriterije, neprimerni pa v izločeni podatkovni tok.

- Povpraševalni podatkovni tok

  Povpraševalni podatkovni tok ne potrebuje usmerjanja in se vedno usmeri v iskalni kriterij, ki je povpraševanje sprožil. Povpraševalni podatkovni tok se preda neposredno filtrirnemu ogrodju.

- Izločeni podatkovni tok

  Izločeni podatkovni tok je rezultat izločanja in se v trenutni implementaciji (zaradi količine podatkov) zavrže.

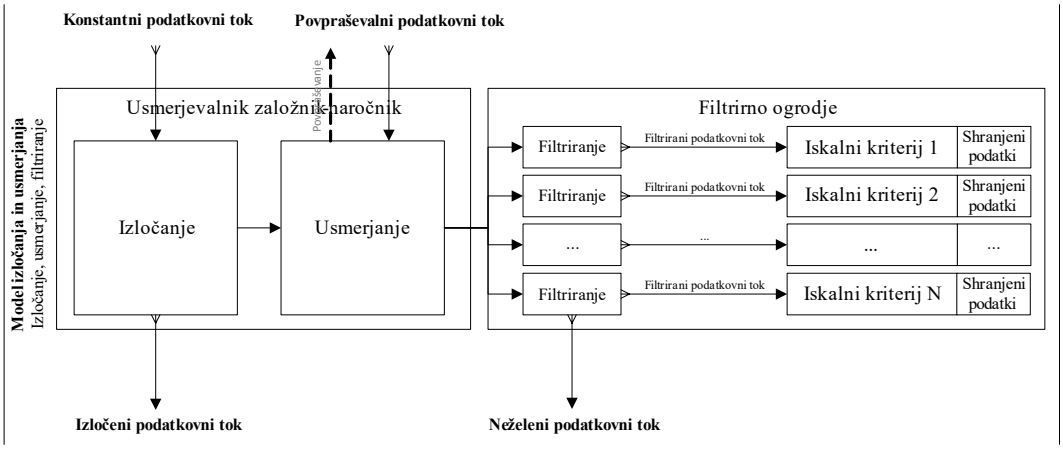
- Neželeni podatkovni tok

  Neželeni podatkovni tok je že usmerjen na prave iskalne kriterije, vendar definicije filtrov preprečujejo shranjevanje njegovih rezultatov. Rezultati v neželenem podatkovnem toku se v trenutni implementaciji shranijo, vendar se ne analizirajo.

- Filtrirani podatkovni tok

  Filtrirani podatkovni tok predstavlja množico podatkov, ki so primerni za analizo in jih obravnava faza analiz, ki je implementirana v modelu za porazdeljevanje bremena.

Shema tokov v povezavi z implementiranima ogrodjema je podana v naslednji sliki:



Slika 5.2: Podatkovni tokovi in ogrodja

Vsi podatki, shranjeni za kasnejšo analizo, se torej nahajajo v iskalnih kriterijih. Z ogrodjem za procesiranje podatkov v modelu za porazdeljevanje bremena (poglavje 6) se podatki analizirajo in kasneje vizualizirajo.

5.2 Izločanje – usmerjevalnik založnik-naročnik

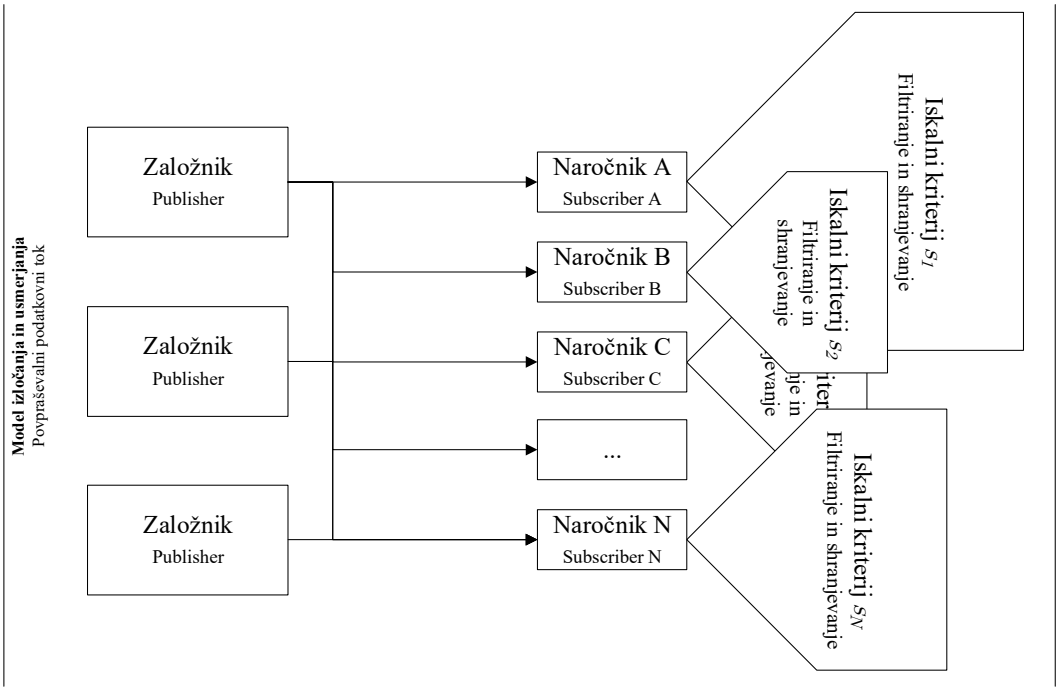
Implementacija izločanja in usmerjanja je izvedena s konceptom založnik-naročnik. V te namene smo implementirali visoko performančni usmerjevalnik, ki je namenjen izmenjavi sporočil med akterji, ki podatke pošiljajo in drugimi odjemalci.

V tem poglavju podajamo način implementacije modela za izločanje in usmerjanje, uporabljene koncepte šibke sklopljenosti in podporo za usmerjanje poljubne vsebine.

Splošna arhitektura mehanizma založnik-naročnik opredeljuje *založnika* (ali več založnikov), ki tipično generirajo *sporočila*. Sporočila se prenašajo na podlagi zunanjih ali notranjih akcij[44]. Obenem v konceptu obstajajo *naročniki*, ki se naročijo na specifične tipe sporočil (imenujemo jih tudi naročniški têrmini). V modelu pričujočega magistrskega dela so založniki ponudniki vsebin na družbenih omrežjih, naročniki iskalni kriteriji, sporočila pa so rezultati, ki se prenašajo od izvora do iskalnega kriterija.

Usmerjevalnik založnik-naročnik tako v jedro procesiranja prejema celoten tok podatkov in izvaja usmerjanje na primerne naročnike – iskalne kriterije, ki se v sistem usmerjanja prijavijo s pogoji (ključnimi besedami). Usmerjevalnik tako usmeri vsak rezultat, ki ustreza kriteriju na pravo mesto.

Spodaj prikazujemo tipično arhitekturo usmerjevalnika založnik-naročnik:



Slika 5.3: Naročniki in založniki

[44] Zunanja akcija je na primer prihod novega rezultata iz toka podatkov, notranja akcija pa dogodek, ki ga sproži sam usmerjevalnik in je namenjen sistemski diagnostiki.

Vsebina, ki se pretaka skozi usmerjevalnik je lahko generična in je predstavljena v obliki tipične storitveno usmerjene arhitekture kot sporočilo. V našem primeru uporabljamo tipizirana sporočila, ki vsebujejo podatke v obliki rezultatov najdenih na družbenih omrežjih. Primer serializiranega sporočila je predstavljen že v konceptualnem modelu, v sliki 2.4 in predstavlja rezultat iz družbenega omrežja Twitter.

Obstajajo naslednji tipi sporočil:

- Netipizirana tekstovna sporočila

  Netipizirana tekstovna sporočila lahko procesira vsak naročnik, ker ne potrebujejo poznati formata oziroma strukture sporočila.

- Tipizirana sporočila

  Tipizirana sporočila imajo obliko klasičnega razreda (serializirane instance razreda), ki je lahko definirana vnaprej. V primeru rezultatov, ki so namenjeni iskalnim kriterijem, je sporočilo rezultata tipizirano in njegov format poznan vsem naročnikom – iskalnim kriterijem. Na voljo so tudi sistemski tipi sporočil, ki se izmenjujejo znotraj sistema in so namenjeni spremljanju delovanja usmerjevalnika.

5.2.1 Naročniški têrmini

Naročniški têrmini[45] so identifikatorji na katere se naročniki prijavijo.

> *Definicija: Naročniški têrmin*
>
> *Naročniški têrmin je identifikator na katerega se naročnik prijavi z namenom prejemanja vseh sporočil, ki jih založniki pošljejo na ta têrmin.*

Primer sistemskega naročniškega têrmina je na primer *!Sys.DispatchService.MessagesReceived*. Têrmin je lahko tudi kateri koli drug tekstovni niz na katerega založniki pošiljajo sporočila.

V primeru uporabe usmerjevalnika v naši problemski domeni so naročniški termini zgoščene vrednosti povpraševalnih nizov.

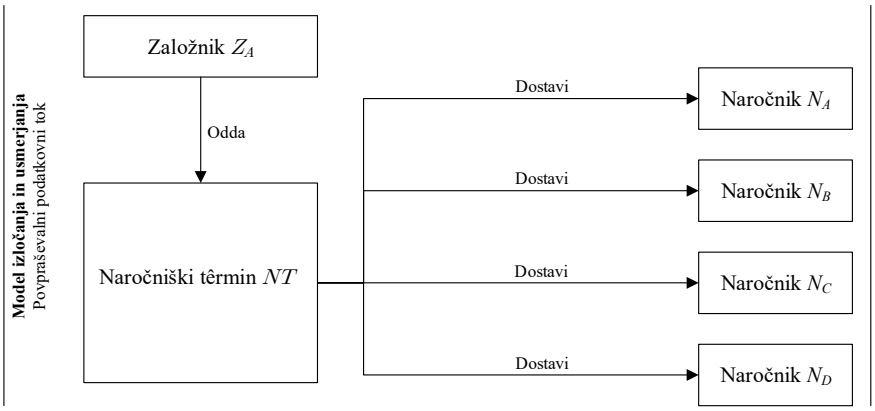
[45] Angleški izraz je *subscription term*.

Definicija: Zgoščena vrednost povpraševalnega niza

Zgoščena vrednost povpraševalnega niza (query string) je zgoščena vrednost SHA1 nad povpraševalnim nizom ali ključno besedo.

Z uporabo zgoščenih vrednosti povpraševalnih nizov je mogoče doseči ujemanje med prihajajočimi rezultati in definicijami iskalnih kriterijev. Iskalni kriteriji se torej naročijo na têrmine, ki jih prej generirajo s SHA1 zgoščevalno funkcijo preko svojih povpraševalnih nizov.

Naslednja shema prikazuje primer dostave sporočil z mehanizmom založnik-naročnik:



Slika 5.4: Dostava sporočil z mehanizmom založnik-naročnik

V zgornjem primeru so naročniki N_A, N_B, N_C in N_D vsi naročeni na naročniški termin NT.

Mehanizem usmerjanja je tako naslednji:

(1) Naročnik N_A se naroči na naročniški têrmin NT_1
(2) Naročnik N_B se naroči na naročniški têrmin NT_1
(3) Naročnik N_C se naroči na naročniški têrmina NT_1 in NT_2
(4) Založnik Z_A odda sporočilo s tekstovno vsebino "Vsebina sporočila A" na naročniški têrmin NT_1
(5) Založnik Z_B odda sporočilo s tekstovno vsebino "Vsebina sporočila B" na naročniški têrmin NT_2
(6) Naročnika N_A in N_B prejmeta tekstovno sporočilo z vsebino "Vsebina sporočila A"
(7) Naročnik N_C prejme dve tekstovni sporočili z vsebinama "Vsebina sporočila A" in "Vsebina sporočila B"

Implementacija usmerjevalnika shranjuje tako podatke o založnikih, kot naročnikih in izvedenih naročilih na naročniške têrmine ter usmerja sporočila za vse naročniške têrmine na vse naročnike.

Sistemski naročniški têrmini imajo posebno sintakso, kar jim omogoča ločevanje od vsebinskih têrminov. Pravila za imenovanje naročniških têrminov so naslednja:

- Têrmini ne razlikujejo med malimi in velikimi črtkami
- Vsebinski têrmini niso sintaktično omejeni, vendar se ne smejo začenjati s klicajem (!)
- Sistemski têrmini se začnejo z nizom *!Sys.**

Za naročnike velja tudi:

- Naročijo se lahko na poljubno število naročniških têrminov

  Vsi naročniki se lahko naročijo na poljubno število têrminov. Teoretično so naročniki lahko tudi založniki, vendar se to v implementaciji opisanega sistema ne uporablja.

- Različni naročniki se lahko naročijo na enake têrmine

  Enake têrmine lahko uporabljamo za razdeljevanje rezultatov več naročnikom. Funkcionalnost je zelo uporabna v primerih ekvivalentnih kriterijev (podrobneje opisano v poglavju 2.3.2, Faza usmerjanja), kjer lahko usmerjevalnik dostavlja enake rezultate več kriterijem, če so za to izpolnjeni pogoji. V primeru dostave več naročnikom smo implementirali paralelno pošiljanje sporočil, ki se izvede hkrati za vse primerne naročnike.

Usmerjanje je torej mogoče izvesti na naslednje načine:

- Od enega založnika na več naročnikov (1 – več)

  Založnik lahko dostavi vsebino na têrmin, na katerega je naročenih več naročnikov. S tem se doseže dostava 1 – več.

- Od več založnikov na enega naročnika (več – 1)

  Več založnikov lahko dostavi vsebino enemu naročniku preko enega ali več têrminov. S tem se doseže dostava več – 1.

- Od več založnikov na več naročnikov (več – več)

Več založnikov lahko dostavi vsebino več naročnikom preko enega ali več têrminov. S tem se doseže dostava več – več.

5.2.2 Arhitektura usmerjevalnika

Osnovna sistemska arhitektura usmerjevalnika je sestavljena iz treh storitev:

- Založniška storitev

  Založniška storitev je potrebna za objavo vsebine. Omogoča objavo netipiziranih in tipiziranih sporočil.

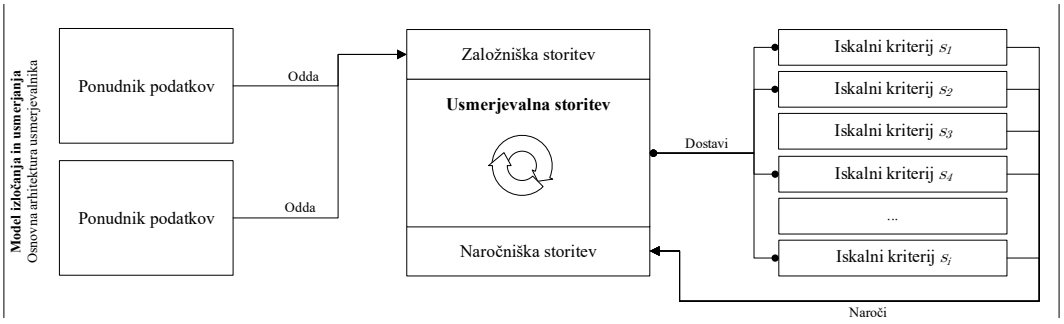
- Naročniška storitev

  Naročniška storitev shranjuje in obvladuje naročnine vseh iskalnih kriterijev.

- Usmerjevalna storitev

  Usmerjevalna storitev implementira pomnilniško (in-memory) usmerjanje sporočil in dostavlja sporočila iz založniške storitve v naročniško storitev in s tem od založnikov k naročnikom.

Naslednji diagram prikazuje osnovno sistemsko arhitekturo usmerjevalnika za izločanje in usmerjanje rezultatov:



Slika 5.5: Sistemska arhitektura usmerjevalnika

5.2.3 Načini dostopa

Načine dostopa do usmerjevalnika založnik-naročnik smo implementirali za sledeče sporočilne izmenjevalne vzorce (message exchange pattern, MEP):

- Potisni (pull)

  Uporablja klasični pub-sub sporočilni izmenjevalni vzorec.

- Potisni / vlečni (push / pull)

  Uporablja klasični sporočilni izmenjevalni vzorec in vzorec zahtevek-odgovor (request-response pattern)

- Vlečni (pull)

  Uporablja samo vzorec zahtevek-odgovor.

Naslednja tabela podaja različne načine dostopa s stališča disperzije med naročnike:

način dostopa	**vzorec**	**disperzija**
potisni	pub-sub	1 – več
potisni / vlečni	pub-sub, request-response	1 – več, 1 – 1
vlečni	request-response	1 – 1

Tabela 5.1: Načini dostopa, izmenjevalni vzorci in disperzija

Opisani načini dostopa do usmerjevalnika omogočajo zanimive razširitve klasičnega pub-sub izmenjevalnega vzorca, ki so posebej uporabne v primeru povpraševalnega podatkovnega toka, specifično:

- Omogoča vsebino na zahtevo (vlečni način)

  Vlečni mehanizem omogoča naročniku pridobivanje zadnjega stanja rezultatov s proženjem eksplicitnega zahtevka. To je posebej priročno za model pridobivanja podatkov v našem primeru, saj implementacija povpraševalnega toka zaradi narave omejitev na strani vmesnikov ponudnikov ne omogočajo uporabe potisnega mehanizma.

- Omogoča zakasnjeno dostavo

  Usmerjevalnik lahko shranjuje sporočila za dostavo ob naslednjem vlečnem klicu. Omogočena je tudi ponovitvena strategija v primerih, ko rezultati niso bili uspešno dostavljeni ali naročnik ni bil dosegljiv.

Razširitve načina dostopa do usmerjevalnika dovoljujejo uporabniku sistema uporabo polnih pub-sub mehanizmov brez komunikacijskih omejitev, ki pri

tem izmenjevalnem vzorcu navadno nastopajo[46]. Implementacija mešanega načina potisni / vlečni omogoča simulacijo polnega pub-sub mehanizma brez potrebe po konstantno odprtih povezavah, kar je izjemno pomembno sploh za mobilne odjemalce, ki se sprehajajo iz popolnoma zaprtih v popolnoma odprta omrežja.

5.3 Vrste filtrov

Implementacija modela vsebuje vključujoče (inkluzivne) in izključujoče (ekskluzivne) filtre. Vključujoči filtri vključujejo – torej spustijo skozi tiste rezultate, ki ustrezajo vsaj eni lastnosti filtra, ostale pa zavržejo. Izključujoči filtri izključijo vse rezultate, ki prekršijo katero koli lastnost filtra.

Implementirali smo naslednje vrste filtrov:

- Jezikovni filter

  Jezikovni filter je namenjen definiciji jezikov katere naj jezikovni detektor uporabi za filtriranje. V primeru, če je rezultat v vsaj enem ustreznem jeziku, ga filter spusti skozi. Filter je inkluziven.

- Uporabniški filter

  Uporabniški filter uporabljamo za filtriranje neželenih uporabnikov. Definicija filtra je tako seznam uporabniških računov, ki jih želimo filtrirati. Filter je ekskluziven.

- Besedni filter

  Besedni filter je mogoče uporabiti za izločanje specifičnih besed, ki jih ne želimo v rezultatih, čeprav celoten rezultat ustreza iskalnemu kriteriju. Filter je ekskluziven.

- Črkovni filter

  Črkovni filtri so namenjeni odstranjevanju neželenih rezultatov, ki so posledica podobnih (ali enakih) ključnih besed v različnih jezikih in omogoča filtracijo nerelevantnih rezultatov. Sestavljen je iz

[46] Potisni mehanizmi zahtevajo prosto komunikacijsko pot od izvora do ponora, kar večkrat predstavlja težavo v primerih, ko je ponor skrit za požarnim zidom.

množice črk, ki v rezultatu ne smejo biti prisotne. Filter je ekskluziven.

- Pametni filter

  Pametni filter je izveden filter, ki ne deluje nad rezultatom samim, pač pa nad profilom uporabnika, ki je rezultat oddal. Z drugimi besedami, pametni filter deluje nad uporabniškim računom in preverja starost profila, število sledilcev, število uporabnikov, ki jim račun sledi, ter število zapisanih mnenj. Filter eliminira rezultate glede na nastavljene minimalne vrednosti[47] in je ekskluziven.

5.3.1 Algoritem filtriranja

Krovni algoritem filtriranja se sprehodi čez vse filtre za vsak rezultat in rezultat izloči takoj, ko je kakšen od filtrov prekršen. Algoritem je implementiran na način, da takoj prekine z delom, čim je prekršena ena od zahtevanih lastnosti. Vrstni red preverjanja je namenoma izveden na način, da se najprej filtrira največji odstotek rezultatov in je tako izvajanje algoritma čim hitrejše.

Vrstni red filtriranja, glede na prej opisane vrste filtrov je:

(1) jezikovni filter
(2) uporabniški filter
(3) besedni filter
(4) črkovni filter
(5) pametni filter

Algoritem filtriranja je naslednji:

```
01    r ← current result
02    languageFiltered ← filter r by language
03    if (languageFiltered) then return
04    userFiltered ← filter r by users
05    if (userFiltered) then return
06    wordFiltered ← filter r by words
07    if (wordFiltered) then return
```

[47] Primer minimalnih vrednosti je: minimalno 5 sledilcev, minimalno 10 uporabnikov, ki jim račun sledi, več kot 60 dni star račun in najmanj 100 objav.

```
08    letterFiltered ← filter r by letters
09    if (letterFiltered) then return
10    smartFiltered ← filter r by smart filter
11    if (smartFiltered) then return
12    save r
```

Algoritem 5.1: Filtriranje

Algoritem v prvi vstici pridobi rezultat, v vrsticah 02 – 11 požene vse filtre in preneha z delom v primeru, če je kakšen od filtrov prekršen. V naprotnem primeru, v vrstici 12, rezultat shrani.

5.3.2 Vključujoči filtri

Psevdo algoritem za vse vključujoče filtre deluje na naslednji način:

```
01    r ← current result
02    for each property p in filter properties
03       if r satisfies p then
04            save r
05            return
06       else
07            continue
08       end if
09    end for
```

Algoritem 5.2: Vključujoči filter

V vrsticah 02 – 09 vključujoči algoritem preveri vse lastnostni inkluzivnega filtra in pri prvi izpolnitvi shrani rezultat – vrstica 04. V nasprotnem primeru nadaljuje z iteriranjem skozi vse lastnosti filtra. Inkluzivno filtriranje se torej zaključi ob prvem ujemanju lastnosti filtra.

5.3.3 Izključujoči filtri

Izključujoči filtri delujejo obratno od vključujočih in rezultat takoj ob prekršitvi lastnosti izločijo.

Algoritem je naslednji:

```
01    r ← current result
02    for each property p in filter properties
03       if r satisfies p then
04            return
05       end if
```

```
06    end for
07    save r
```

Algoritem 5.3: Izključujoči filter

Algoritem za izključujoče filtriranje preveri lastnosti rezultata v vrsticah 02 – 06 in takoj prekine z delom, ko je izpolnjena prva lastnost (vrstica 04). V nasprotnem primeru nadaljuje s preverjanjem vseh lastnosti in rezultat shrani šele na koncu (vrstica 07).

Pametni filtri so izključujoči, zato je njihov psevdo algoritem podoben algoritmu 5.3, pri čemer velja:

```
01    r ← current result
02    minAge ← minimum account age
03    minPosts ← minimum number of posts age
04    minFollowers ← minimum followers
05    minFollowing ← minimum following
06    if age of account of r < minAge then
07       return
08    end if
09    if number of posts of account of r < minPosts then
10       return
11    end if
12    if number of followers of account of r < minFollowers then
13       return
14    end if
15    if following accounts of account of r < minFollowing then
16       return
17    end if
18    save r
```

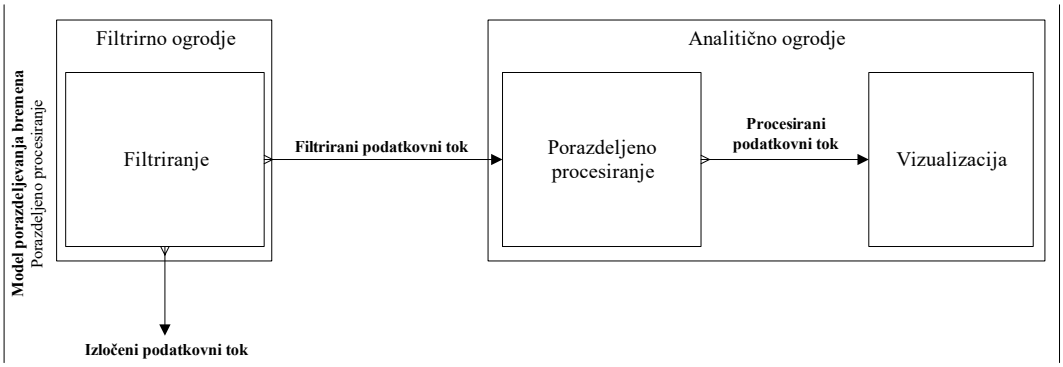
Algoritem 5.4: Pametni filter

Algoritem se od 5.3 razlikuje po vnaprej znanem številu lastnosti, ki se preverijo na računu, ki je oddal rezultat r.

6 Model porazdeljevanja bremena

V poglavju podajamo implementacijo modela za porazdeljevanje računskega bremena, ki nastaja iz zahtev za analize podatkov filtriranega podatkovnega toka.

S stališča podatkovnih tokov je shema procesiranja modela naslednja:



Slika 6.1: Porazdeljeno procesiranje in podatkovni tokovi

Implementacija porazdeljenega procesiranja ima dve operaciji:

(1) Razdeljevanje dela

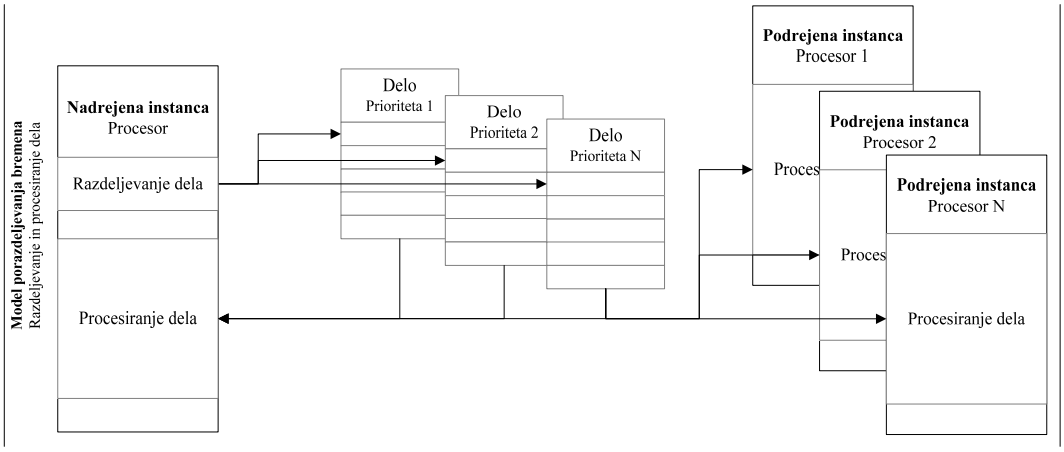
Razdeljevanje dela je operacija, ki definira katero delo mora biti opravljeno. Razdeljevanje opravlja nadrejena instanca, ki določi katere analize in za katera časovna okna je potrebno opraviti izračun.

(2) Procesiranje dela

Procesiranje dela opravljajo vse instance, vključno z nadrejeno instanco, kadar ta ne razdeljuje dela. Operacija, ki se izvede, je odvisna od definicije dela (vrste analize).

Vhodni podatki za analize so vsi podatki, ki so bili zajeti z modelom za pridobivanje podatkov. Izhodni podatki predstavljajo rezultate analiz.

Na naslednji sliki je prikazan način razdeljevanja dela med nadrejenimi in podrejenimi instancami:



Slika 6.2: Razdeljevanje in procesiranje dela

Nadrejene procesorske instance delo razdeljujejo med podrejene instance z uporabo sporočilnih vrst (message queues). Definicija potrebnega dela je podana v obliki sporočila in zapisana v sporočilno vrsto. Vrste imajo prioritete, glede na naravo dela, ki ga je potrebno opraviti. V sistemu torej obstaja koncept prioritete dela (work priority), ki se manifestira v obliki prioritetne vrste dela.

Definicija: Prioritetna vrsta dela

Prioritetna vrsta dela je sporočilna vrsta, ki vsebuje delo določene prioritete. V sistemu obstajajo štiri prioritete (prioriteta 1 – prioriteta 4), glede na analize, ki jih je potrebno opraviti.

Nadrejena instanca, ki jo izvolimo z uporabo elekcijskega algoritma (poglavje 6.1), se odloča kakšno prioriteto bo delu dodelila glede na okoliščine razdeljevanja dela. V primeru, če oddaja delo za analize v realnem času, mu bo dodelila delo 1, če ponavlja razdeljevanje dela, pa prioriteto 4.

Prioritete dela so:

- 1. prioriteta: analize v realnem času

  Predstavlja delo najvišje prioritete, katerega vse instance procesorjev najprej procesirajo. Prioriteta je namenjena procesiranju analiz v realnem času[48].

[48] Izvajajo se med pridobivanjem podatkov, v kvazi realnem času, za vsa časovna okna.

- 2. prioriteta: rekalkulacije obstoječih analiz

  Predstavlja delo druge prioritete, ki ga instance procesirajo samo in le v primeru, če nimajo dela prioritete 1. Prioriteta je namenjena procesiranju rekalkulacij obstoječih analiz[49].

- 3. prioriteta: zgodovinske analize

  Predstavlja delo tretje prioritete, ki ga instance procesirajo samo in le v primeru, če nimajo dela prioritete 2. Prioriteta je namenjena procesiranju zgodovinskih analiz[50].

- 4. prioriteta: ponovljene analize

  Predstavlja delo četrte prioritete, ki ga instance procesirajo samo in le v primeru, če nimajo dela prioritete 3. Prioriteta je namenjena procesiranju analiz, ki so se neuspešno izvedle[51] ali se niso izvedle zaradi napak v času procesiranja.

Princip delovanja v točki razdeljevanja dela je torej naslednji: nadrejena instanca preveri kakšno delo je potrebno opraviti in delo razdeli po ustreznih prioritetah. Podrejene instance delo nemudoma opravijo, pri čemer se breme enakomerno porazdeli med vse delujoče instance. Vedno se najprej opravi delo z najvišjo prioriteto (prioriteta 1) in nadaljuje z nižjimi prioritetami, dokler vse oddano delo ni opravljeno. V primeru, če podrejene instance nimajo dela, počakajo dokler nadrejena instanca dela ne razdeli.

6.1 Vrste vlog

V implementaciji modela obstajajo tri vrste vlog, ki opravljajo različno delo:

(1) Iskalna vloga

Vloga, ki opravlja iskanja (povpraševanja) novih rezultatov. Vloga je sestavljena iz množice instanc, kjer je ena instanca nadrejena instanca,

[49] Rekalkulacije obstoječih analiz se izvajajo v primerih, ko je prišlo do brisanja podatkov, spremembe analitičnih podatkov ali spremembe parametrov za izračun analiz. Tipičen primer spremembe parametrov je sprememba uporabljenega modela sentimenta.

[50] Analize, ki se izvajajo za pretekla obdobja. Ob kreiranju novega iskalnega kriterija se izvedejo analize za obdobje od časa prvega rezultata do trenutnega časa.

[51] Analize, katerih procesiranje ni uspelo

ostale so podrejene instance. Porazdeljevanje bremena se izvaja z mehanizmom nadrejenih-podrejenih instanc. Vlogo smo v konceptualnem modelu opisali v poglavju 2.2, njena implementacija pa je podana v poglavju 4.2. Nadrejena instanca te vloge se vóli in mora biti vedno prisotna.

(2) Procesorska vloga

Vloga, ki opravlja procesiranje analiz in je sestavljena iz množice instanc, kjer je ena instanca nadrejena, ostale pa so podrejene instance. Porazdeljevanje dela se izvaja z mehanizmom nadrejenih-podrejenih instanc. Nadrejena instanca te vloge se vóli in mora biti vedno prisotna.

(3) Spletna vloga

Spletna vloga streže uporabniški vmesnik rešitve in je sestavljena iz množice enakovrednih (peer) instanc. Porazdeljevanje bremena (streženja spletnih strani uporabniškega vmesnika) se izvaja z omrežnim porazdeljevanjem bremena[52].

6.1.1 Iskalna vloga – nadrejena instanca

Namen nadrejene instance iskalne vloge je opravljanje dela, ki ga ni smiselno paralelizirati med vse podrejene instance. Iskalna nadrejena instanca tudi ne opravlja razdeljevanja dela, saj se podrejene instance z zaklepanjem iskalnih kriterijev (poglavje 4.2.3) same dogovorijo kako bo iskalno delo porazdeljeno.

Nadrejena instanca zato opravlja naslednje delo:

(1) Upravlja z urniki iskalnih kriterijev

Urniki iskalnih kriterijev omogočajo spreminjanje prioritete iskanja in frekvence analiz za posamezne iskalne kriterije za točno določena časovna obdobja (navadno so to obdobja ob širše zanimivih dogodkih). Instanca poskrbi, da se prioriteta iskalnega kriterija ustrezno poveča/zmanjša, kar neposredno vpliva na izbor iskalnega kriterija v fazi povpraševanj in na frekvenco analiz, v fazi deljenja dela nadrejene instance procesorske vloge.

(2) Skrajšuje sistemski dnevnik

[52] Omrežno porazdeljevanje bremena (network load balancing), ki deluje na podlagi afinitete odjemalčevega naslova IP omogoča, da ista strežniška instanca znotraj spletne seje vedno streže istemu odjemalcu.

Nadrejena instanca skrajšuje sistemski dnevnik z namenom ohranjanja obvladljivega števila zapisov v njem.

(3) Opravlja asinhroni izračun sentimenta

Nadrejena instanca opravlja tudi asinhroni izračun sentimenta za vse že shranjene rezultate pri katerih iz neznanega vzroka izračun sentimenta ni uspel v času pridobivanja rezultatov. Ker je izračun sentimenta (analiza sentimenta) procesno draga operacija, lahko v času pridobivanja rezultatov pride do napake iz vzroka časovne omejitve, nedostopnosti vira ipd., zato nadrejena instanca na določeno periodo preveri obstoj takih primerov in jih uredi.

6.1.2 Iskalna vloga – podrejena instanca

Podrejene instance iskalne vloge opravljajo samo eno nalogo: pridobivanje novih podatkov iskalnih kriterijev. Proces pridobivanja podatkov je podan v poglavju 4.2. Vse podrejene instance delujejo na enak način in hkrati periodično preverjajo obstoj nadrejene instance.

6.1.3 Procesorska vloga – nadrejena instanca

Nadrejena instanca procesorske vloge razdeljuje procesorsko zahtevno delo – definicije izračuna analiz, ki jih morajo opraviti podrejene instance.

Definicijo dela določa:

- Iskalni kriterij

  Vsaka definicija dela vsebuje iskalni kriterij nad katerim je potrebno analizo izvesti.

- Vrsta analize

  Vrsta analize, ki bo izvedena s strani podrejene instance.

- Prioriteta dela

  Prioriteta dela določa kako pomembno je delo glede na prioritetno shemo.

- Obdobje analize

  Časovno obdobje analize določa čas za katerega je analiza izvedena.

> Privzeto nadrejena instanca nalaga delo za vsa časovna okna, za vse iskalne kriterije.

Delo se deli hkrati za vse omogočene iskalne kriterije, kar pomeni, da v eni periodi razdeljevanja dela v prioritetne vrste vpišemo N_p zahtevkov za analize, kjer je $N_p = w \times t \times s$, in w število časovnih oken, t število vrst analiz in s število iskalnih kriterijev.

Naslednja tabela prikazuje primer zahtevkov za izračun analiz, ki jih nadrejena instanca razdeli v eni periodi.

iskalni kriterij	**vrsta analize**	**prioriteta**	**od**	**do**
1	HC	1	2016-06-29 10:00	2016-06-29 10:05
1	RC	1	2016-06-29 10:00	2016-06-29 10:05
...	...	...	...	...
1	SC	1	2016-06-29 10:00	2016-06-29 10:05
2	HC	1	2016-06-29 10:00	2016-06-29 10:05
2	RC	1	2016-06-29 10:00	2016-06-29 10:05
...	...	...	...	...
2	SC	1	2016-06-29 10:00	2016-06-29 10:05
...	...	...	...	...
n	HC	1	2016-06-29 10:00	2016-06-29 10:05
n	RC	1	2016-06-29 10:00	2016-06-29 10:05
...	...	...	...	...
n	SC	1	2016-06-29 10:00	2016-06-29 10:05

Tabela 6.1: Primer zahtevkov analiz – razdeljevanje dela

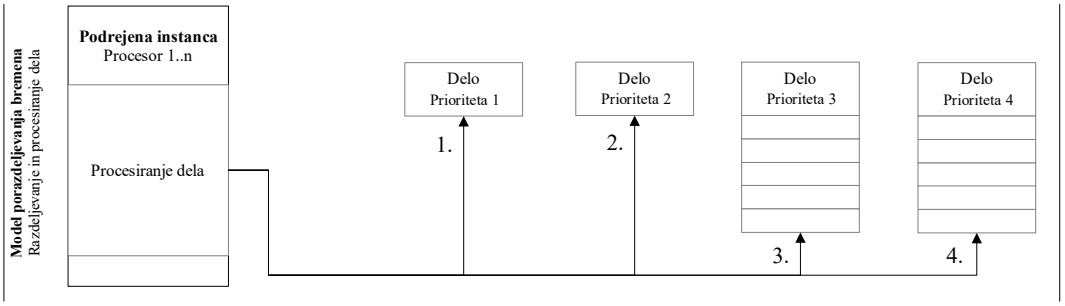
Zgornji primer prikazuje zapise, ki jih nadrejena instanca razdeli v prioritetno vrsto 1 (analize v realnem času) za vse iskalne kriterije (1 – n), za tipe analiz HC (število zadetkov – hit count), RC (doseg – reach count) in SC (analiza sentimenta – sentiment count) ter za pet-minutno časovno okno od 2016-06-29 10:00 do 2016-06-29 10:05. Očitno je, da je v vsaki periodi razdeljevanja takih zapisov mnogo več.

Točneje, nadrejena instanca v prioritetno vrsto uvrsti identifikator do definicije dela, ki je zapisan v relacijski podatkovni bazi. Podrejene instance nato preko identifikatorja preberejo opis dela (descriptor), kjer je podana

vrsta analize, obdobje analize ter dodatni meta podatki, ki določajo ali gre za sistemsko delo (v nasprotju z ročnim, uporabniško iniciiranim zahtevkom) ter časom zahtevka, planiranim časom izvajanja analize in dejanskim časom izvajanja analize.

6.1.4 Procesorska vloga – podrejena instanca

Podrejene instance procesorske vloge opravljajo samo eno nalogo – procesiranje analiz. Vsaka instanca svoje lahko procesira katero koli delo, ne glede na prioriteto ali iskalni kriterij, ki mu pripada. Vrstni red procesiranja podrejenih instanc je prikazan v naslednji sliki.



Slika 6.3: Vrstni red procesiranja podrejenih instanc

Na zgornji sliki podrejene instance procesirajo delo prioritete 3, ker višje prioritete v času procesiranja nimajo razdeljenega dela. Vrstni red procesiranja je vedno od višje prioritete (1) do nižje prioritete (4).

Podrejena instanca po procesiranju dela v relacijsko tabelo zapiše rezultate dela, vpiše dejanski čas opravljanja analize ter delo (sporočilo) izvrsti iz vrste.

6.2 Procesni model instanc

Procesni model vseh instanc, ne glede na vrsto vloge je enak. Instanca se po zagonu vedno najprej inicializira, prebere osnovne podatke za delovanje, pregleda stanje svoje vloge (nadrejena, podrejena) in nadaljuje z delom, ki je na voljo.

Splošen pregled delovanja instance od zagona naprej je naslednji:

```
01    id ← instance identifier
02    type ← instance type
```

```
03    isMaster ← false
04    shouldRun ← true
05    initialize instance
06    while (shouldRun)
07       if (instance enabled) then
08             if (isMaster) then
09                   // do master work according to type
10             else if (!isMaster or
11                         (isMaster and masterIsProcessor))
12                   // do slave work according to type
13             end if
14       else
15             if (instance should stop) then
16                   shouldRun ← false
17             end if
18             sleep a little
19       end if
20       log iteration for instance identifier id
21    end while
```

Algoritem 6.1: Procesni model instanc

Procesni model v vrsticah 01 – 05 inicializira instanco. V glavni zanki se izvajanje nadaljuje s procesiranjem vse dokler ni eksplicitno prekinjeno. Vsaka vloga instance ima možnost onemogočitve[53], zato instanca ob zagonu preveri ali je omogočena (vrstica 07). Sledi preverjanje katero delo naj instanca opravlja – vlogo nadrejene instance, vlogo podrejene instance ali mešano, v kateri tudi nadrejene instance izvajajo procesiranje. Sledi izvajanje glede na vlogo (vrstice 08 – 13) in preverjanje ali naj instanca zaključi z izvajanjem (vrstice 15 – 17). Po zaključku vsake iteracije, vse instance vpišejo opravljeno delo v dnevnik sistema (vrstica 20).

[53] Instance se v času razvoja in odprave napak izvajajo tudi v lokalnih razvojnih okoljih in potrebujejo možnost onemogočitve predvsem zaradi odražanja realnega stanja in sodelovanja instanc v oblaku. V času razvoja se take instance onemogočijo zato, da ne vplivajo na izvajanje testne ali produkcijske platforme.

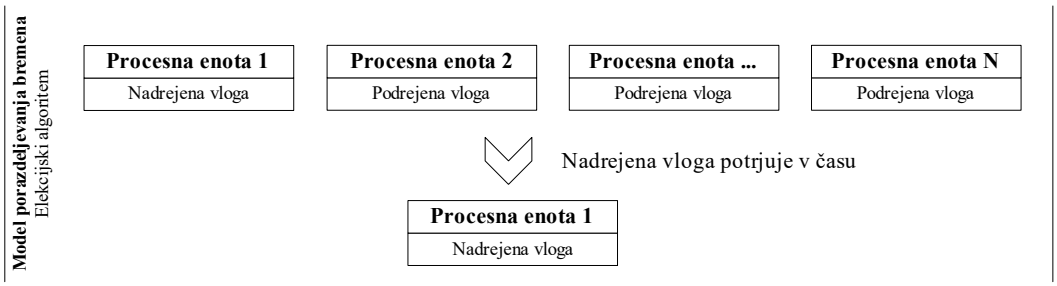
6.3 Elekcijski algoritem

Elekcijski algoritem zagotavlja vzpostavitev nadrejene instance med vsemi trenutno delujočimi instancami enake vloge. Vsaka iskalna in procesna vloga ima svojo nadrejeno instanco.

Pomembnost nadrejene instance je velika. Instanca vedno nadzoruje količino dela podrejenih instanc, hkrati pa opravlja neparalelizabilne operacije ali operacije za katere bi bilo nesmotrno vpeljevati paralelizacijo[54]. Nadrejena instanca mora v implementiranem modelu vedno obstajati, zato njen obstoj periodično preverja tudi vsaka podrejena instanca.

Nadrejena instanca svoje pravilno delovanje in obstoj potrjuje vsako periodo. V primeru, če nadrejena instanca ne potrdi svojega pravilnega delovanja v času, ki je znan vsem instancam, se zgodi *reelekcija*. Reelekcija je proces, kjer delujoče podrejene instance izberejo novo nadrejeno instanco. Do reelekcije pride samo in le v primeru, ko nadrejena instanca ne uspe potrditi svojega delovanja znotraj časovne periode, ki je za to določena.

Skladno s sliko 2.15 (poglavje 2.4.1), v naslednjih shemah prikazujemo proces reelekcije glede na implementirani algoritem.



Slika 6.4: Proces reelekcije – nadrejena vloga potrjuje

V primeru, ko nadrejena vloga potrjuje svoj obstoj in razdeljevanje dela znotraj vnaprej znane periode, do reelekcije ne pride.

Vzroki za reelekcijo so lahko:

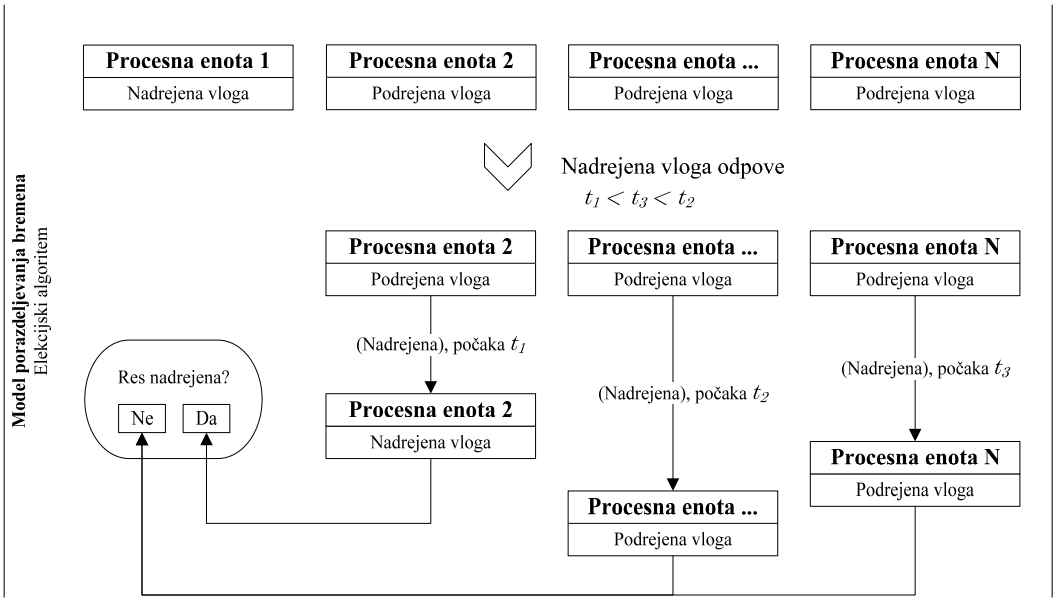
- Splošna odpoved delovanja nadrejene instance
- Odpoved povezljivosti nadrejene instance

[54] Predvsem režijske operacije kot so čiščenje dnevnika, upravljanje z urnikom, pošiljanje obvestil, itd.

- Zasedenost nadrejene instance z drugim režijskim delom

Ne glede na vzrok odpovedi nadrejene instance se s procesom reelekcije ena od obstoječih podrejenih instanc spremeni v nadrejeno instanco, ta pa, ko prične z delovanjem, v običajno podrejeno instanco. Ne glede na trenutno vlogo instance je torej način delovanja katere koli instance po izvedeni reelekciji lahko obrnjen – podrejena instanca lahko postane nadrejena in obratno.

V primeru odpovedi nadrejene instance pride do reelekcije. Proces je naslednji:



Slika 6.5: Proces reelekcije – odpoved nadrejene instance

Elekcijski algoritem je naslednji:

```
01    masterInstanceId ← get current master instance identifier
02    instanceId ← current instance identifier
03    isMasterInstance ← false
04    // check if master instance exists
05    if (masterInstanceId does not exist) then
06       randomTime ← random time delay
07       wait randomTime
08       masterInstanceId ← get current master instance
                            identifier
09       if (masterInstanceId does not exist) then
10            // this instance should be master from now on
11            masterInstanceId ← instanceId
```

```
12            isMasterInstance ← true
13            return
14       end if
15    else
16       if (instanceId == masterInstanceId) then
17            isMasterInstance ← true
18            extend master instance for instanceId
19            return
20       else
21            masterTimely ← is masterInstanceId timely
22            if (masterTimely) then
23                 // master instance is timely -> leave as is
24                 isMasterInstance ← false
25                 return
26            else
27                 // master instance is not timely -> elect
28                 randomTime ← random time delay
29                 wait randomTime
30                 masterInstanceId ← get current master
                                      instance identifier
31                 if (instanceId == masterInstanceId) then
32                      // reelection succeeded
33                      isMasterInstance ← true
34                      return
35                 else
36                      // another instance took over
                        // back off
37                      isMasterInstance ← false
38                 end if
39            end if
40       end if
41    end if
```

Algoritem 6.2: Elekcijski algoritem

Algoritem v vrstici 01 pridobi trenuten identifikator nadrejene instance, v vrstici 02 pa identifikator trenutne instance. V primeru, da nadrejena instanca sploh ne obstaja (vrstica 04) algoritem neposredno skoči v fazo elekcije. V nasprotnem primeru (vrstice 16 – 19) algoritem preveri ali je nadrejena instanca enaka trenutni instanci in če je, podaljša njeno veljavnost. Če nadrejena instanca ni trenutna instanca (druga instanca je

nadrejena instanca) algoritem preveri njeno potrjevanje delovanja. V primeru, če druga nadrejena instanca ne zamuja s potrjevanjem (je točna), algoritem ne izbere nove nadrejene instance (vrstice 22 – 25). V nasprotnem primeru pride do reelekcije, kot je prikazano v vrsticah 27 – 38.

6.4 Sporočilne vrste

Sporočilne vrste v modelu porazdeljevanja bremena opravljajo ključno vlogo pri razdeljevanju dela. Nadrejena instanca definicijo dela zapisuje v prioritetne sporočilne vrste, podrejene instance pa delo procesirajo na podlagi pristopa prvi-pride-prvi-melje.

V poenostavljenem primeru, kjer model uporablja samo eno prioriteto za vse vrste dela, velja naslednje:

- Nadrejena instanca delo vpisuje v vrsto vsako časovno periodo (poglavje 2.4.2)
- Podrejen instance[55] delo procesirajo tako, da iz vrste vzamejo definicijo dela in delo opravijo

Ob vsaki uvrstitvi definicije dela v vrsto se vrsta podaljša. Ob vsakem procesiranju dela se vrsta skrajša. Dolžina vrste tako vedno nakazuje količino dela, ki ga je potrebno opraviti oziroma dela, ki je že definirano, vendar še ni bilo opravljeno.

Uvrščanje (enqueuing) in izvrščanje (dequeuing) sta izvedena transakcijsko. Uvrščanje v eni transakcijski operaciji vloži v vrsto vse definicije dela, ki so potrebne za dano časovno periodo in iskalni kriterij, za katerega nadrejena instanca razdeljuje delo. Podobno se izvrščanje izvaja z nastavljivo stopnjo paralelizma, kjer podrejene instance iz vrste vzamejo N sporočil hkrati in jih vzporedno procesirajo.

Recimo, da je stopnja paralelizma za procesne enote (podrejene instance, ki opravljajo delo preko porazdeljevanja bremena) enaka deset (10) in trenutno v sistemu deluje pet (5) podrejenih instanc, nadrejena instanca pa ne opravlja dela.

[55] Model predvideva možnost procesiranja tudi s strani master instanc.

Primer uvrščanja in izvrščanja je potem naslednji:

(1) Nadrejena instanca uvrsti 100 definicij dela v trenutni periodi
(2) Podrejena instanca 1 nemudoma izvrsti 10 zahtevkov za delo
(3) Podrejena instanca 2 nemudoma izvrsti 10 zahtevkov za delo
(4) Podrejena instanca 3 nemudoma izvrsti 10 zahtevkov za delo
(5) Podrejena instanca 4 nemudoma izvrsti 10 zahtevkov za delo
(6) Podrejena instanca 5 nemudoma izvrsti 10 zahtevkov za delo
(7) Podrejena instanca 1 opravi delo
(8) Podrejena instanca 2 opravi delo
(9) Podrejena instanca 3 opravi delo
(10) Podrejena instanca 4 opravi delo
(11) Podrejena instanca 5 opravi delo
(12) Podrejena instanca 1 izvrsti naslednjih 10 zahtevkov za delo
(13) Podrejena instanca 2 izvrsti naslednjih 10 zahtevkov za delo
(14) Podrejena instanca 3 izvrsti naslednjih 10 zahtevkov za delo
(15) Podrejena instanca 4 izvrsti naslednjih 10 zahtevkov za delo
(16) Podrejena instanca 5 izvrsti naslednjih 10 zahtevkov za delo
(17) Podrejena instanca 1 opravi delo
(18) Podrejena instanca 2 opravi delo
(19) Podrejena instanca 3 opravi delo
(20) Podrejena instanca 4 opravi delo
(21) Podrejena instanca 5 opravi delo

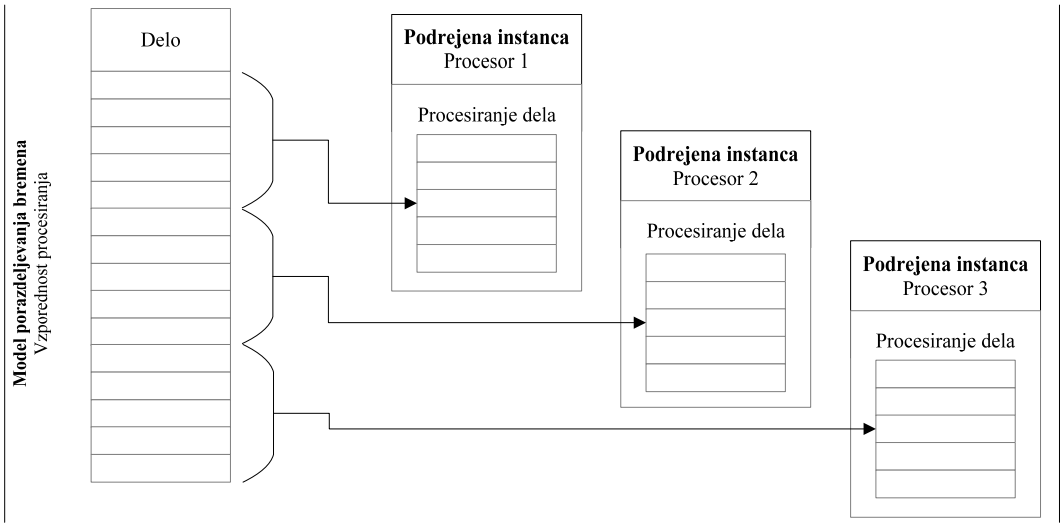
Opisan primer prikazuje časovna obdobja, kjer je dolžina vrste 100, 50 in 0. Takoj po točki (1) je dolžina vrste 100, po točki (11) je enaka 50 in ob končanju procesiranja, po točki (21) je vrsta prazna.

V opisanem primeru je zaporedje procesiranja zaradi lažjega prikaza enako zaporedju pridobivanja definicij dela, kar v realnem primeru ni nujno tako. Zaradi različnih časov trajanja analiz instance redko procesirajo dodeljeno delo v vrstnem redu pridobivanja.

Vzporednost procesiranja se tako odvija na dveh nivojih:

(1) Znotraj posamezne podrejene instance – glede na določeno stopnjo paralelizma
(2) Na več instancah, ki vzporedno tečejo v oblaku

Spodnja shema prikazuje vzporednost procesiranja glede na vrsto za definicijo dela.



Slika 6.6: Vzporednost procesiranja

Na shemi je prikazano vzporedno procesiranje 15 enot dela s strani treh (3) podrejenih instanc s stopnjo paralelizma pet (5).

6.4.1 Samodejno skaliranje

V modelu smo implementirali samodejno skaliranje na podlagi dolžine vrst, ki vsebujejo definicije dela. V primeru, ko se vrste podaljšajo, sistem samodejno poveča število instanc, ki sodelujejo v modelu porazdeljevanja bremena in jih analogno zmanjša, ko se vrsta skrajša.

Glede števila instanc in samodejnega skaliranja veljajo naslednje omejitve:

(1) Minimalno število instanc je dve (2)
Minimalno število se zahteva zaradi predpostavke, da mora v modelu vedno obstajati vsaj ena nadrejena in ena podrejena instanca, ne glede na nastavitev, kjer lahko nadrejena instanca opravlja tudi procesiranje.

(2) Število instanc se povečuje in zmanjšuje linearno s skupno dolžino prioritetnih vrst, ki vsebujejo definicije dela
Število potrebnih instanc v danem trenutku se izračuna iz skupne dolžine vrst, ki vsebujejo definicije dela. Število potrebnih instanc je $N = M/K$, kjer je M skupna dolžina vrst, K pa povprečna zmogljivost procesiranja

posamezne instance. Glede na konstantno posodabljanje zmogljivost instanc ponudnikov procesiranja v oblaku se K posodablja vsakič, ko pride do povečanja zmogljivosti s strani ponudnika v oblaku.

(3) Maksimalno število instanc je nastavljivo
Maksimalno število instanc je zaradi preventivnih razlogov prevelike porabe vseeno omejeno in odvisno od poslovnega modela naročnika.

6.5 Rezultati testiranja

Porazdeljevanje bremena smo testirali na realnem, živem delovnem bremenu (workload). Specifično smo se orientirali na naslednje karakteristike:

(1) Število opravljenih analiz na eno (1) minuto za naslednje vrste analiz: analiza števila zadetkov (AZ), analiza dosega (AD), analiza sentimenta (AS)
(2) Število povpraševanj za nove podatke na eno (1) minuto

Meritve smo izvajali glede na število podrejenih instanc, pri čemer smo bili posebej pozorni, da v času testiranja ni prihajalo do izpada nadrejene instance in je bilo delo vedno točno in pravilno razdeljeno. V primeru, če je zaradi napake ali nepredvidenega izpada prišlo do izpada nadrejene instance, smo testiranje ponovili.

Testni pogoji so bile sestavljeni iz:

- 1000 iskalnih kriterijev z različnimi povpraševalnimi nizi, različnimi filtri in različno frekvenco rezultatov. 200 iskalnih kriterijev je imelo najnižjo prioriteto (1), 200 nizko prioriteto (2), 200 normalno prioriteto (3), 200 višjo prioriteto (4) in 200 najvišjo prioriteto (5).
- 2 – 100 iskalnih enot (instanc iskalne vloge), kjer je bila ena (1) instanca vedno nadrejena
- 2 – 100 procesnih enot (instanc procesorske vloge), kjer je bila ena (1) instanca vedno nadrejena
- Vse instance so se izvajale v enem podatkovnem centru ponudnika storitev v oblaku, konkretno Microsoft Azure – North Europe Data Center.

Metrika testiranja procesorske vloge je bila:

$$n_P = n_{AZ} + n_{AD} + n_{AS} \ [/min] \tag{6.1}$$

Metrika podaja število opravljenih analiz n_P, kot vsoto števila analiz števila zadetkov n_{AZ}, analiz dosega n_{AD} in analiz sentimenta n_{AS}.

Metrika testiranja iskalne vloge je bila strukturirana podobno, vendar smo se pri njej osredotočali le na število izvedenih iskanj. Metrika iskalne vloge, n_I, je bila:

$$n_I = n_{i1} + n_{i2} + n_{i3} + n_{i4} + n_{i5} \ [/min] \tag{6.2}$$

Števila n_{i1} do n_{i5} predstavljajo izvedeno število iskanj prioritete 1 do prioritete 5[56].

Heterogenost smo dodatno zagotavljali s testiranjem na realnih povpraševalnih nizih, ki so posledica uporabe dejanskih, produkcijskih iskalnih kriterijih uporabnikov.

6.5.1 Testiranje procesorske vloge

Skladno z enačbo (2.41) je minimalno število analiz pri 200 kriterijih prioritete 5 enako 33.600/min. S padanjem prioritete (in enakem številu kriterijev za vsako prioriteto, 200) se zmanjšuje tudi potrebno število analiz, torej: prioriteta 4: 6.720/min, prioriteta 3: 2.240, prioriteta 2: 1.120/min in prioriteta 1: 560/min.

Skupno je moral sistem torej izračunati najmanj 44.240 analiz vsako minuto, oziroma okoli 740 analiz vsako sekundo.

V spodnji tabeli podajamo rezultate testiranja procesorske vloge.

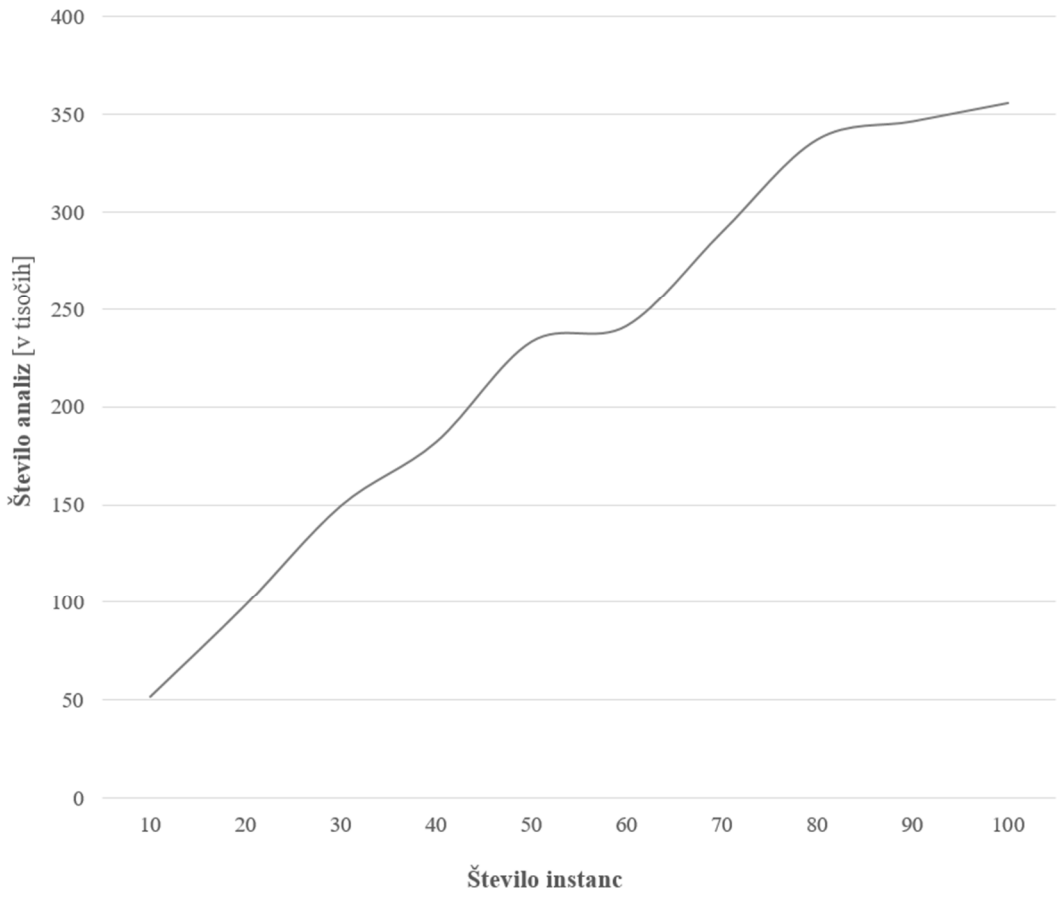
št. instanc	n_P	n_{AZ}	n_{AD}	n_{AS}
2	9.228	3.968	3.230	2.030
3	13.889	5.694	4.306	3.889
4	18.118	7.972	5.798	4.348
5	23.890	9.795	7.167	6.928
6	29.388	12.931	9.698	6.759
7	38.389	17.275	12.284	8.829

[56] Prioriteta 1 je najnižja, prioriteta 5 je najvišja.

8	43.009	18.494	15.053	9.462
9	**47.837**	**20.092**	**13.873**	**13.873**
10	51.419	22.110	15.940	13.369
20	98.388	38.371	30.500	29.516
30	149.338	64.215	49.282	35.841
40	181.911	80.041	65.488	36.382
50	233.188	97.939	74.620	60.629
60	241.388	98.969	84.486	57.933
70	289.778	121.707	98.525	69.547
80	337.229	145.008	111.286	80.935
90	346.499	152.460	117.810	76.230
100	355.901	163.714	121.006	71.180

Tabela 6.2: Testiranje procesorske vloge

Kritično točko procesiranja smo dosegli z devet (9) procesorskimi instancami, od katerih jih je vseh devet opravljalo delo (tudi nadrejena instanca).



Slika 6.7: Linearnost skaliranja procesorske vloge

V tabeli 6.1 je kritična točka procesiranja označena **krepko**. Slika 6.7 prikazuje grafični pogled na vpliv skaliranja na število opravljenih analiz. Rezultati jasno kažejo na linearni trend povečevanja števila opravljenih analiz na minuto. Skupno opravljeno število analiz n_P, se po pričakovanjih in napovedih modela povečuje skoraj linearno.

Sploščenje grafa v področju nad 80 instanc avtorji pripisujejo možnosti distribucije večjega števila instanc v različne fizične strežniške omare podatkovnega centra. Ponudnik storitve v oblaku ne zagotavlja izvajanja vseh instanc storitve v isti fizični omari (zaradi lokacijske bližine) in dovoljuje distribucijo na poljubne lokacije znotraj podatkovnega centra.

6.5.2 Testiranje iskalne vloge

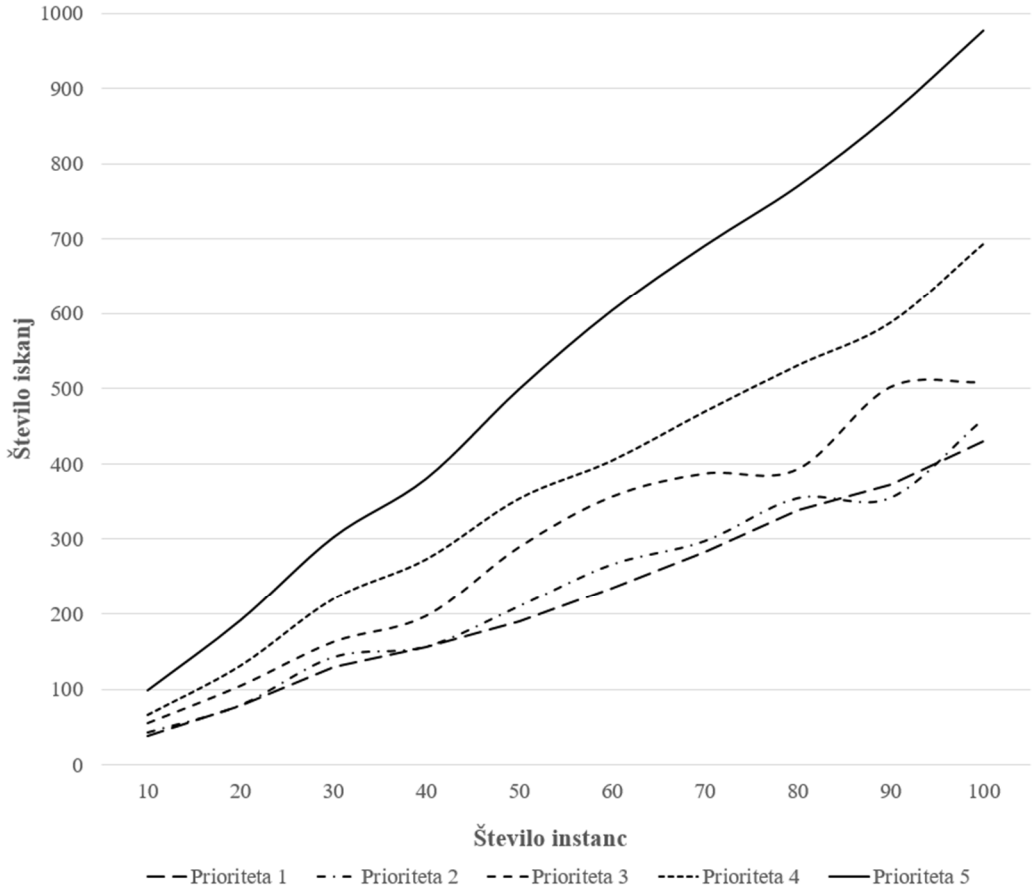
Pri testiranju iskalne vloge smo preverjali število iskanj za iskalne kriterije različnih prioritet. Rezultati so podani v spodnji tabeli.

št. instanc	n_I	n_{i1}	n_{i2}	n_{i3}	n_{i4}	n_{i5}
2	62	8	9	11	14	20
3	96	13	14	16	22	31
4	126	18	19	21	27	41
5	155	21	21	26	38	50
6	191	28	23	33	42	65
7	221	31	31	37	50	72
8	247	35	34	42	56	80
9	275	33	40	49	64	88
10	301	38	42	56	67	98
20	584	78	78	105	132	191
30	957	130	142	163	220	302
40	1.163	156	156	198	274	380
50	1.542	190	210	289	354	499
60	1.866	236	266	356	405	604
70	2.128	283	297	387	470	691
80	2.387	339	354	393	531	770
90	2.682	372	355	502	588	865
100	3.068	430	459	508	694	977

Tabela 6.3: Testiranje iskalne vloge

Rezultati v tabeli kažejo število sproženih iskanj (in pridobivanja rezultatov) na minuto glede na različno število uporabljenih instanc.

Prioritetno uravnavanje skaliranja iskanja deluje skladno s pričakovanji, saj velja $n_{i1} \leq n_{i2} \leq n_{i3} \leq n_{i4} \leq n_{i5}$ v vseh izmerjenih primerih. Iskalni kriteriji z višjo prioriteto (prioriteta 5) pridobijo več računskih virok in posledično opravijo več povpraševanj.



Slika 6.8: Linearnost skaliranja iskalne vloge

Z gledišča skupnega števila opravljenih iskanj n_I, dosegamo linearno povečanje – odvisno od števila instanc, ki opravljajo iskalno vlogo. Rezultati prikazujejo skupno število opravljenih iskanj na minuto za vseh pet prioritet iskalnih kriterijev.

7 Model vizualizacije pridobljenih analiz

V poglavju podajamo implementacijo modela vizualizacije pridobljenih analiz, ki skrbi za grafični prikaz znotraj analitičnega ogrodja.

Implementacija modela je ločena glede na naravo analiz, ki se lahko predprocesirajo (procesirani podatkovni tok) in čakajo na zahtevo po prikazu s strani uporabnika in analiz na zahtevo, ki jih uporabnik izvaja neposredno med uporabo sistema.

S stališča modela za vizualizacijo analize ločimo na dva osnovna tipa:

(1) Predprocesirane analize

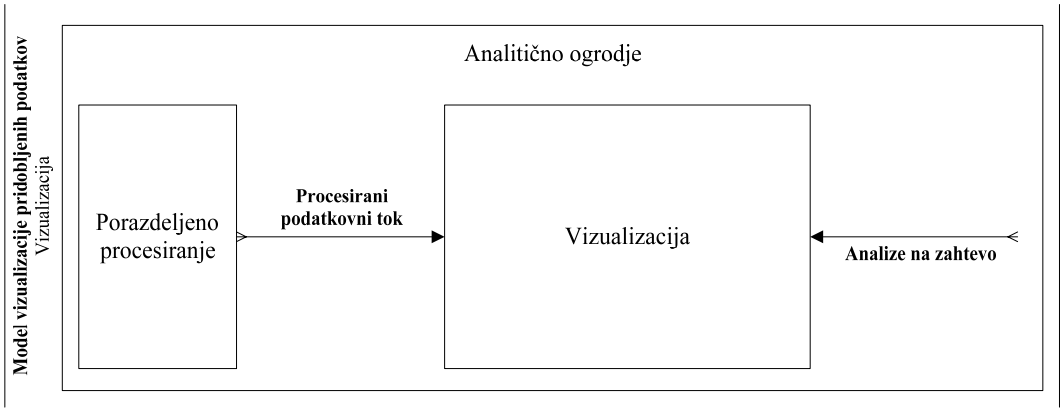
Podatki za predprocesirane analize so izračunani vnaprej. Analize so izračunane za tipična časovna obdobja in časovna okna, ki jih definira model za porazdeljevanje bremena. Med predprocesirane analize štejemo tudi analize, katerim uporabnik nastavi poljubno časovno obdobje, vendar se izračunavajo asinhrono in so tako s stališča vizualizacije tudi predprocesirane.

(2) Analize na zahtevo

Analize na zahtevo se izračunajo sinhrono, takoj po zahtevku za prikaz s strani uporabnika. Tipično so to analize za katere so potrebna natančna časovna obdobja, ki so odvisna od zajetih podatkov in jih uporabnik določi z uporabno časovnega selektorja glede na rezultate drugih analiz. Izračunavanje takih analiz vnaprej bi bilo nesmiselno[57].

[57] Tipičen primer je padec sentimenta po nekem dogodku. Uporabnik padec identificira na predprocesirani analizi sentimenta in se poglobi v besedni oblak, ki se izračuna sinhrono.

S stališča uvrstitve modela v sistem je prikaz analiz zadnji v vrsti razvitih modelov znotraj tega dela. V sistem ga umeščamo tako, kot kaže naslednja shema.



Slika 7.1: Umestitev modela za vizualizacije pridobljenih analiz

7.1 Vrste analiz

Trenutna implementacija modela za vizualizacijo vsebuje naslednje vrste analiz:

(1) Analiza števila zadetkov
Podaja število zadetkov glede na definiran iskalni kriterij skozi čas.

(2) Analiza dosega
Podaja maksimalni potencialen doseg uporabnikov, ki govorijo o vsebini, ki jo ujame določeni iskalni kriterij.

(3) Analiza sentimenta
Podaja absolutni in relativni sentiment najdene vsebine glede na definirani iskalni kriterij.

(4) Analiza vsebine
Podaja resonanco vsebine ali kako močno je vsebina odmevala v družbenih omrežjih.

(5) Analiza generatorjev vsebine
Podaja število generatorjev vsebine (uporabnikov ali računov) in število različnih lokacij iz katerih je bila vsebina generirana.

(6) Analiza najaktivnejših uporabnikov
Podaja najaktivnejše uporabnike ali račune, ki so omenjali vsebino, ki jo pokriva iskalni kriterij.

(7) Analiza jezikov

Podaja največkrat uporabljene jezike pri vsebini posameznega iskalnega kriterija.

(8) Analiza besednih oblakov

Podaja enobesedne, dvobesedne in trobesedne besedne oblake najdene vsebine.

(9) Percepcijska analiza

Podaja percepcijsko analizo (tudi percepcijsko mapo ali zaznavalno analizo) vsebine iskalnega kriterija.

(10) Analiza oblaka značk

Podaja pogostost uporabljenih značk (tags) v najdeni vsebini iskalnega kriterija.

(11) Analiza oblaka komunikacije

Podaja pogostost komunikacije med različnimi uporabniki ali računi v najdeni vsebini iskalnega kriterija.

Ker je sistem *neodvisen od vrste analiz*[58] in podrobnosti implementacije posamezne analize presegajo obseg tega dela, v naslednjih poglavjih podajamo opise nekaj najbolj tipičnih analiz, ki so bile uporabljene za prikaz delovanja modela in smiselnost sistema za procesiranje, analizo in vizualizacijo podatkov z mehanizmi visoke skalabilnosti.

7.1.1 Analiza števila zadetkov

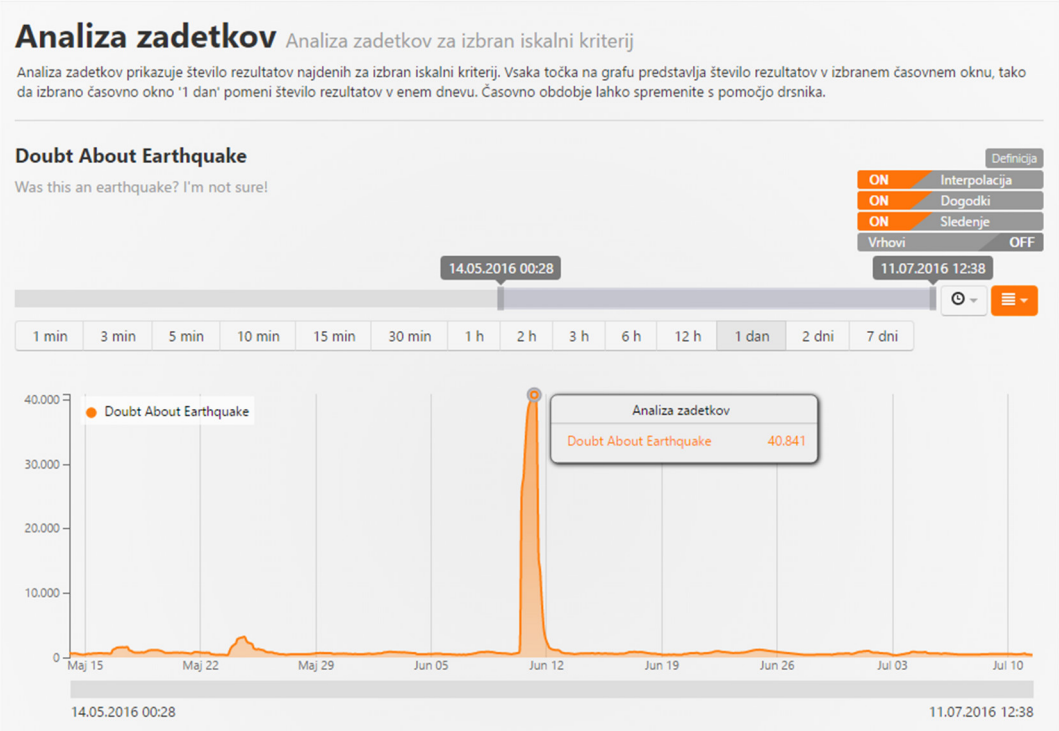
Analiza števila zadetkov je ena od najpomembnejših in hkrati najpreprostejših analiz. Podaja število najdenih zadetkov iskalnega kriterija v različnih časovnih obdobjih in za različna časovna okna.

Število zadetkov seveda prikazuje vključenost širše javnosti v specifično témo. Podaja tudi zanimanje za določeno témo in v primerih večjih skokov je mogoče sklepati, da je množica različnih uporabnikov pričela z govorjenjem o določeni témi zaradi dogodka, ki je vplival na širšo javnost. Analiza se uporablja tudi za ugotavljanje zanimanja javnosti za določeno témo.

[58] Sistem porazdeljevanja bremena lahko izračunava poljubne analize in jih prikazuje v podprtih načinih vizualizacije.

Število zadetkov je definirano kot skupno število zadetkov, ki jih ujame povpraševalni niz iskalnega kriterija v nekem časovnem obdobju. Ker gre za predprocesirano analizo, so obdobja vedno enaka časovnim oknom in si sledijo s periodo izvajanja analiz (poglavji 2.4.2 in 2.4.3).

Primer analize števila zadetkov podajamo v spodnji sliki:



Slika 7.2: Analiza števila zadetkov – dnevno časovno okno

Slika prikazuje kriterij *Dvom o potresu (Doubt About Earthquake)*, ki je zasnovan tako, da spremlja vse objave, kjer je uporabljena beseda *earthquake*, prisoten vprašaj (*?*) in ima vključen jezikovni filter angleščine. Na ta način sistem zajame vse angleške rezultate, ki omenijo potres in se ob tem sprašujejo ali je do njega zares prišlo. Špica v analizi zadetkov jasno prikazuje verjeten čas zaznave potresa s strani javnosti[59].

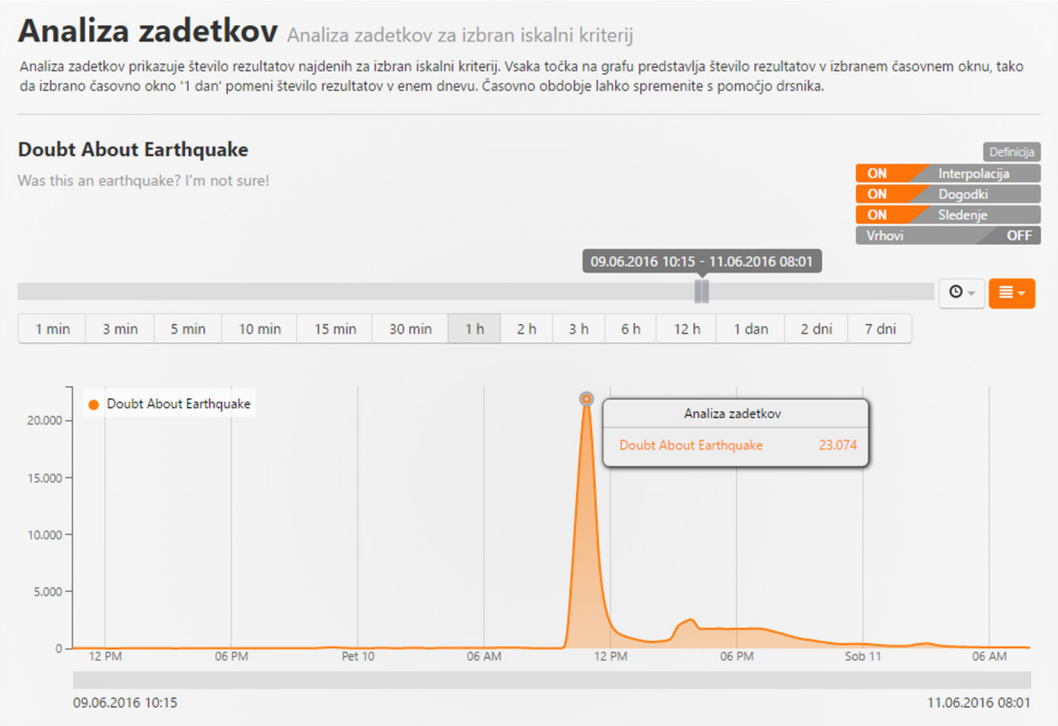
Iz očitnega skoka v številu zadetkov je mogoče sklepati, da je šlo za nenavaden dogodek, saj je šum v primeru takega kriterija na zelo nizkem nivoju (600 – 800 zadetkov dnevno). V dotičnem primeru je potres generiral preko

[59] Potres magnitude 5.2 v južni Kaliforniji, v bližini San Diega, sredo, 8.6.2016, http://edition.cnn.com/2016/06/10/us/earthquake-san-diego-california-irpt/

40.000 zadetkov, če upoštevamo dnevni vrh – časovno okno je v zgornji sliki nastavljeno na 1 dan.

Za točnejši čas dogodka je mogoče pogledati analize z manjšimi časovnimi okni, zato uporabnik navadno zoži pogled (izvede povečevanje) in se osredotoči na manjša časovna okna od enega dneva.

Spodnja slika prikazuje isti dogodek z uporabo eno urnega (1h) časovnega okna.



Slika 7.3: Analiza števila zadetkov – urno časovno okno

Podrobnejši pogled tako prikazuje tudi točnejši čas (začetek 11:00 CEST, vrh 11:30 CEST), kar se ujema z uradnim časom potresa (01:04:39 AM PST)

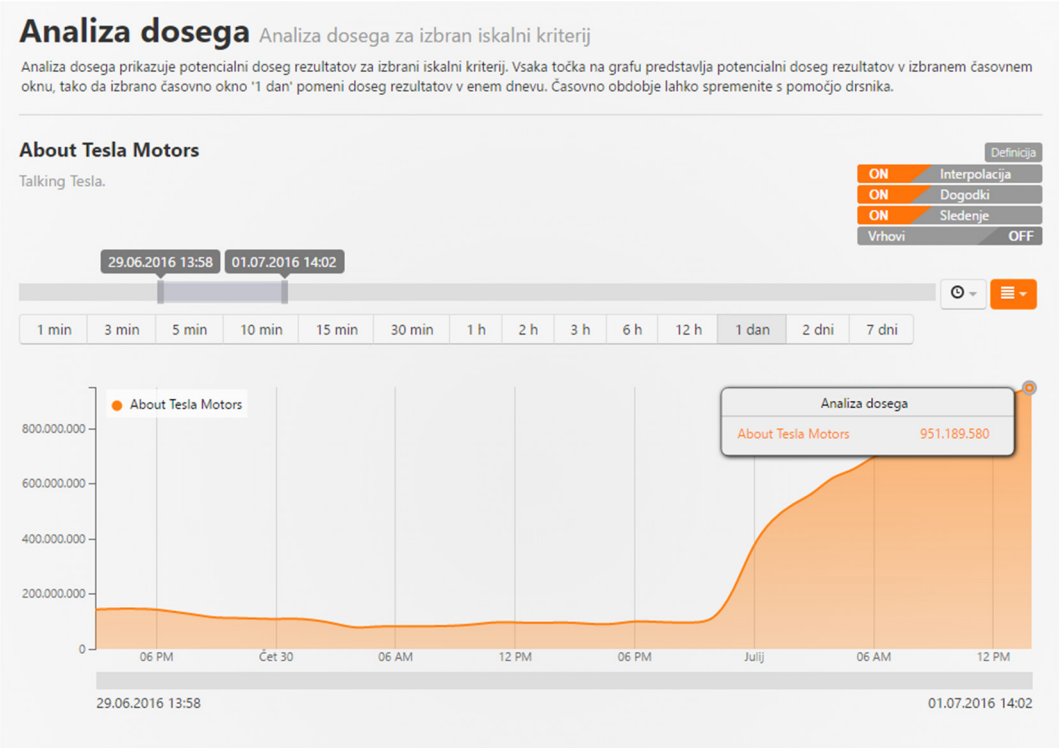
7.1.2 Analiza dosega

Analiza dosega podaja *maksimalni potencialni doseg* vsebine, ki jo zajame iskalni kriterij.

> *Definicija: Maksimalni potencialni doseg*
>
> *Maksimalni potencialni doseg je vsota vseh sledilcev vseh uporabnikov, ki oddajo vsebino, ki se ujame v iskalni kriterij. Maksimalni potencialni doseg bi bil dosežen samo in le v primeru, če bi vsi sledilci vseh uporabnikov, ki oddajo vsebino, to vsebino tudi prebrali.*

Doseg prikazuje vidnost vsebine v družbenih omrežjih in določa hitrost razširjanja. Večji kot je doseg, večja je verjetnost, da je več uporabnikov vsebino zaznalo. Doseg se hitro povečuje v primeru retvitanja ali deljenja vsebin na družbenih omrežjih, kadar to izvajajo uporabniki z velikim številom sledilcev.

Primer analize dosega je prikazan spodaj.



Slika 7.4: Analiza dosega

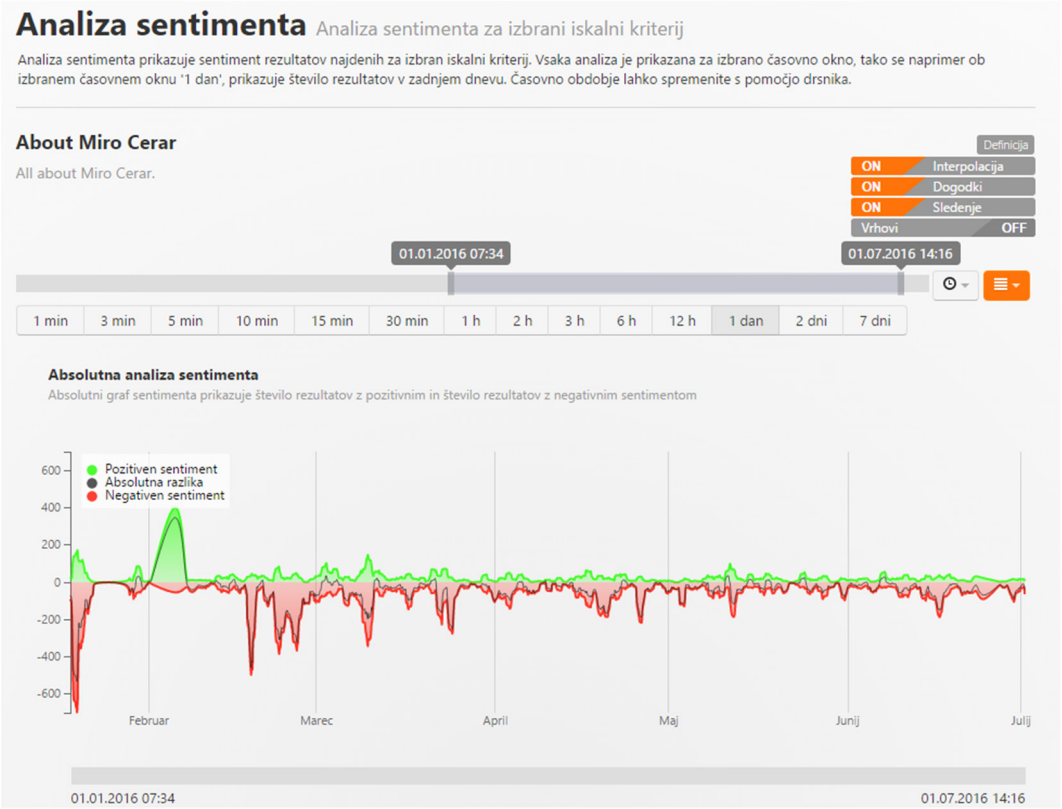
Prikazana je analiza dosega za podjetje Tesla Motors v obdobju od 29.6.2016 do 1.7.2016. Očitno je v tem intervalu prišlo do večjega zanimanja širše javnosti ali pa do večje aktivnosti mnenjskih voditeljev. V primeru korelacije z analizo števila zadetkov, uporabniki sistema lahko ugotovijo vzroke za povečano zanimanje pri omenjenem kriteriju.

7.1.3 Analiza sentimenta

Analiza sentimenta je v primeru ugotavljanja mnenja širše javnosti največkrat uporabljena analiza. Rezultati analize predstavljajo čustveno reakcijo, ki jo sistem pridobi z analizo posamezne tekstovne vsebine, ki ustreza iskalnemu kriteriju.

Namen tega magistrskega dela ni raziskava ali razvoj pristopov za ugotavljanje sentimenta iz podatkov na družbenih omrežjih, čeprav je avtor sodeloval pri več projektih (na primer [25]), kjer smo opisan sistem za porazdeljevanje bremena uporabljali skupaj z raziskovalno skupino na Institutu Jožef Stefan in analizo sentimenta, ki smo jo trenirali z modelom SVM (Support Vector Machines).

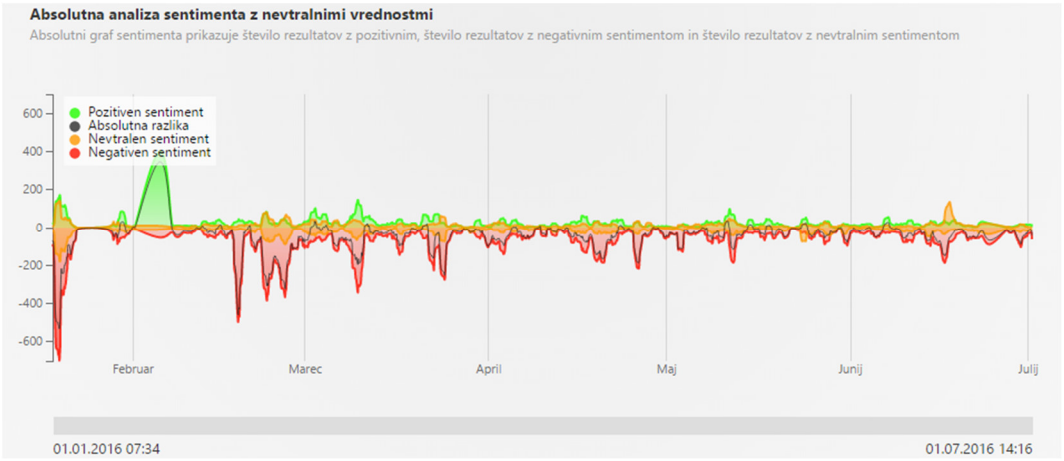
Analiza sentimenta prikazuje absolutno število najdenih pozitivnih sentimentov, negativnih sentimentov in njihovo absolutno razliko.



Slika 7.5: Absolutna analiza sentimenta

Zgornja slika prikazuje ugotovljeni sentiment na primeru Predsednika vlade Republike Slovenije, dr. Mira Cerarja za obdobje od 1.1.2016 do 1.7.2016.

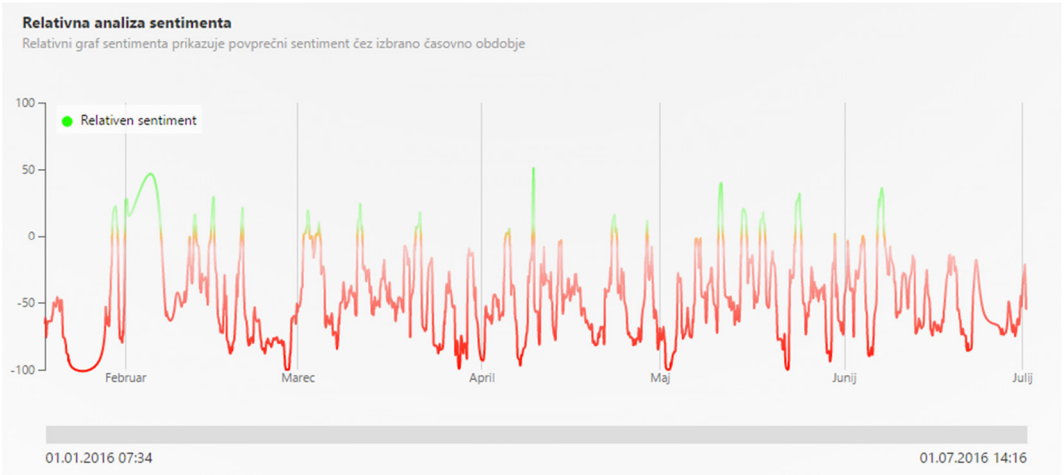
V primeru, če v analizo vključimo tudi nevtralno cono SVM modela (prikazana oranžno), sistem prikaže naslednjo analizo sentimenta.



Slika 7.6: Absolutna analiza sentimenta z nevtralno cono

Sistem vključuje tudi relativno analizo sentimenta, kjer v časovnem oknu za relativni sentiment vzamemo povprečje najdenih pozitivnih in negativnih sentimentov v najdeni vsebini. Relativni sentiment lahko v primerih z manjšim številom rezultatov kaže zelo izkrivljeno sliko, saj je lahko strogo negativen ali strogo pozitiven tudi v primeru samo nekaj rezultatov, ki so vsi polarizirani enako.

Analiza relativnega sentimenta, ki ne upošteva števila zadetkov, je za isti primer (dr. Miro Cerar, 1.1.2016 – 1.7 2016) prikazana spodaj.



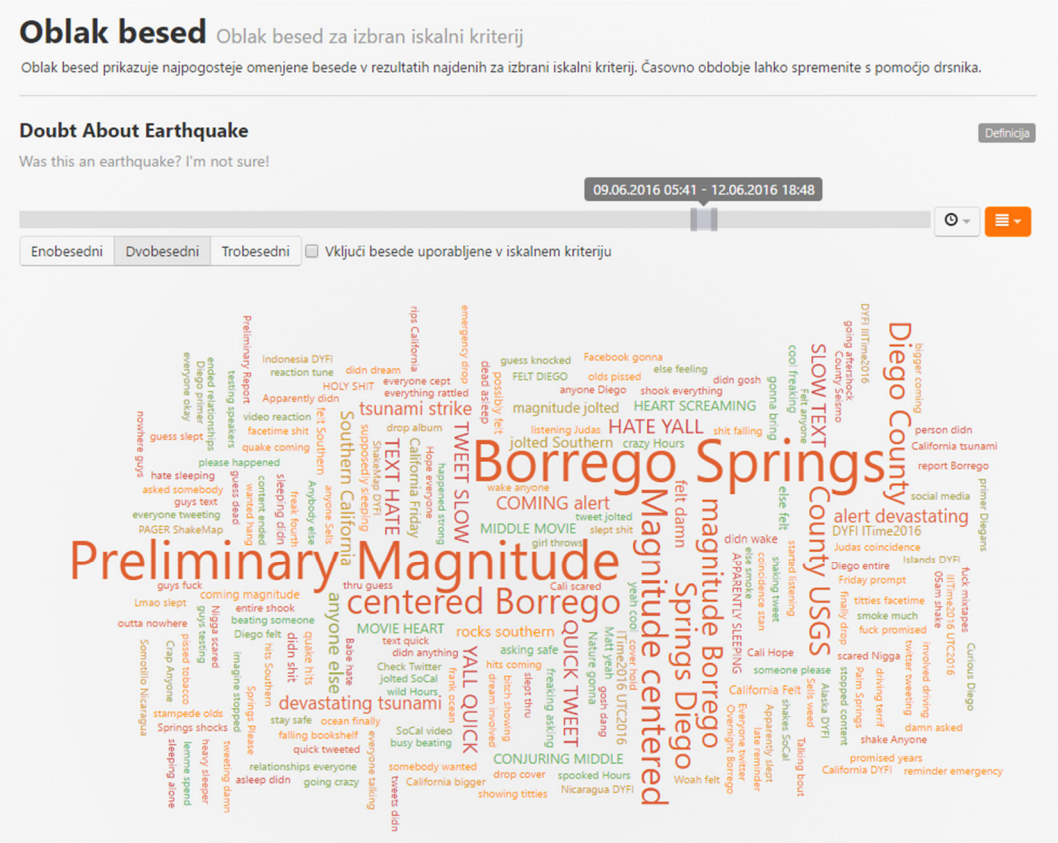
Slika 7.7: Relativna analiza sentimenta

7.1.4 Analiza besednih oblakov

Analiza besednih oblakov prikazuje pogostost uporabljenih besed, dvobesednih in trobesednih zvez v povezavi z iskalnim kriterijem. Implementacija besednega oblaka z barvami hkrati prikazuje tudi sentiment konteksta, torej

sentiment celotne vsebine, v kateri je bila uporabljena beseda ali besedna zveza.

Če se vrnemo na kriterij *Dvom o potresu* in analizo zaženemo nad obdobjem v času potresa v južni Kaliforniji, dobimo naslednji prikaz.



Slika 7.8: Analiza besednih oblakov

Slika prikazuje najbolj uporabljene besedne zveze v času potresa. Očitno je večina uporabnikov, ki je potres čutila ali o njem poročala, v svoji vsebini omenila tudi področje (Borrego Springs, severovzhodno od San Diega, San Diego County). Sentiment kontekstualne vsebine je bil zelo negativen, na kar kaže večinsko rdeče obarvan besedni oblak.

7.1.5 Percepcijska analiza

Percepcijska analiza prikazuje unigrame, bigrame ali trigrame na dvodi-menzionalni percepcijski ravnini, kjer ena dimenzija pomeni frekvenco (pojavnost), druga pa sentiment. Osnovni podatki za to analizo so pridobljeni iz podatkov besednih oblakov, izrisana pa je v prikazu, ki je predvsem uporabljen v družboslovni stroki.

Spodnja slika prikazuje percepcijsko analizo za primer Predsednika vlade Republike Slovenije, dr. Mira Cerarja v času stavke in ustavitve delovanja Luke Koper (27.6.2016 – 11.7.2016).

Slika 7.9: Percepcijska analiza

8 Sklepne ugotovitve

V tem magistrskem delu preučujemo:

- Model za pridobivanje podatkov iz tokovnih izvorov, ki je združen s povpraševalnim načinom pridobivanja podatkov
- Model filtriranja pridobljenih podatkov, ki ustreza tako tokovnim, kot povpraševalnim mehanizmom
- Model za masovno procesiranje poljubnih operacij, ki se izvajajo v oblaku in omogočajo visoko stopnje paralelizacije tako znotraj samih procesorskih enot, kot med njimi
- Model vizualizacij za napredne preglede in vizualizacije analiz, ki so posledica procesiranja vhodnih podatkov

Razviti model pridobivanja podatkov predstavlja generično osnovo za pridobivanje poljubnih podatkovnih struktur iz internetnih ali lokalnih virov, ki v sisteme prihajajo preko tokov ali povpraševanj. V primerih širokih tokov model predvideva uporabo mehanizmov založnik-naročnik z namenom eliminacije podatkov, ki za problemsko domeno niso primerni. Široki tokovi so obravnavani s posebno skrbnostjo, saj predstavljajo tako performančni, kot velikostni problem v smislu procesiranja podatkov in njihovega shranjevanja.

Povpraševalni tok, ki omogoča selektivnost virov, natančnejšo obravnavo in *filtriranje pri izvoru*, se izkazuje kot nepogrešljiv del modela, saj ponuja možnosti, ki jih samo s tokovnimi izvori težko implementiramo. Združevanje podatkov obeh tokov v enoten sistem filtriranja in usmerjanja pa implementatorjem omogoča obvladovanje enotne točke, preko katere podatki

prehajajo, kar dodatno posplošuje opisani sistem za porazdeljeno procesiranje, analizo in vizualizacijo podatkov.

Opisani model izločanja, usmerjanja in filtriranja predstavlja ločljiv del sistema in ga je mogoče uporabiti za poljubne podatkovne izvore in ponore. V našem primeru vanj usmerjamo rezultate modela za pridobivanje podatkov, pri čemer potrebno poudariti, da je model agnostičen na število filtrov in tipe filtrov, hkrati pa omogoča vtično arhitekturo tako na vhodu, kot izhodu.

Model porazdeljevanja bremena, ki ponuja možnosti porazdeljenega procesiranja na več računskih enotah vzporedno, je podan v obliki odpornega, skalabilnega in elastičnega mehanizma za izračunavanje kompleksnih operacij v oblaku. Model je mogoče prenesti na poljubnega ponudnika z viri izračunavanja in ga uporabiti za napade na komplekse probleme, kjer je v kratkem času potrebna velika računska moč. Predstavljeni model podaja mehanizme za razdeljevanje dela, porazdeljevanje izračunavanja med sodelujoče procesne enote in shranjevanje rezultatov. Hkrati model podaja načine za spremljanje zasedenosti sistema s samodejnim načinom skaliranja, ki je odvisen od trenutne količine dela na katerega reagira s samodejnim povečevanjem in zmanjševanjem števila procesorskih enot.

V modelu za vizualizacije pridobljenih analiz se osredotočamo predvsem na primernost predstavitve podatkov analiz, ki v dotičnem primeru opisujejo problemsko domeno. Razviti kontrolniki uporabniku omogočajo hitro navigacijo, učinkovito in natančno obvladovanje časovnih intervalov ter predvsem izbiro načinov prikaza, ki so za dotične analize primerni in pričakovani s strani uporabnikov. Podajamo tudi načine za vizualizacijo pridobljenih podatkov, ki omogočajo intuitivno in hitro brskanje med analizami, njihovimi časovnicami in časovnimi okni.

8.1 Možne izboljšave

Možna izboljšava modela za pridobivanje podatkov vključuje pametnejše obvladovanje izločenega podatkovnega toka, ki se v trenutnem modelu neposredno zavrže. Enako velja za neželeni podatkovni tok, kjer je problem

količinsko sicer veliko manjši, vsebuje pa več potencialno uporabnih rezultatov. Razmišljanje gre predvsem v smer inteligentne zaznave katero podmnožico podatkov bi bilo smiselno shraniti v obeh primerih. Glede na količino izločenega podatkovnega toka, je shranjevanje celotnega toka trenutno ekonomsko nesmiselno. Pristop v smeri samodejnega shranjevanja rezultatov, ki si sicer izločeni, vendar shranjeni v primeru samodejnega prevoda ključnih besed iskalnih kriterijev za vnaprej znane jezike, pa se izkazuje za obetajoč.

Model porazdeljevanja bremena je mogoče razširiti s podporo za več nadrejenih instanc (multi-master), kar bi prineslo še večjo odpornost na napake in manjše periode izpada v fazi razdeljevanja dela. Trenutno model podpira eno nadrejeno instanco za vsako vlogo znotraj konceptualnega modela, zato bi bila razširitev dobrodošla tako za podporo modela pridobivanja podatkov, kot porazdeljevanja bremena.

Mehanizem samodejnega skaliranja (povečevanje števila instanc) je mogoče dopolniti z inteligentnim povprečenjem in napovedovanjem skokov, s čimer se avtorji niso ukvarjali. Pristop bi omogočal naprednejše in prilagodljivejše skaliranje v primerih sunkovitih skokov in s tem večjo elastičnost sistema v takih robnih primerih. Hkrati pa bi to doprineslo tudi k hitrejšem zmanjševanju potreb po virih, kar bi pozitivno vplivalo na cenovni pogled gostovanja take rešitve v oblaku.

9 Seznam uporabljenih virov

[1] J. Gubbia, R. Buyyab, S. Marušić and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Elsevier Future Generation Computer Systems,* vol. 29, no. 7, p. 1645–1660, 2013.

[2] I. Hashema, I. Yaqooba, N. Anuara, S. Mokhtara, A. Gania and S. U. Khan, "The Rise of Big Data on Cloud Computing: Review and Open Research Issues," *Elsevier Information Systems,* vol. 47, p. 98–115, 2015.

[3] S. Kaisler, F. Armour, J. Espinosa and W. Money, "Big Data: Issues and Challenges Moving Forward," in *Hawaii International Conference on System Sciences (HICSS)*, Wailea, Maui, 2013.

[4] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand and Y. Robert, "Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms," *IEEE Transactions on Parallel and Distributed Systems,* vol. 15, no. 4, pp. 319 - 330, 2004.

[5] M. A. Shah and D. Kulkarni, "Storm Pub-Sub: High Performance, Scalable Content Based Event Matching System Using Storm," *Parallel and Distributed Processing Symposium Workshop (IPDPSW),* p. 585–590, maj 2015.

[6] M. Xu, L. Cui, H. Wang and Y. Bi, "A Multiple QoS Constrained Scheduling Strategy of Multiple Workflows for Cloud Computing,"

IEEE International Symposium on Parallel and Distributed Processing with Applications, pp. 629-634, avgust 2009.

[7] Z. Hou, Y. Huang, S. Zheng, X. Dong and B. Wang, "Design and Implementation of Heartbeat in Multi-machine Environment," in *Conference on Advanced Information Networking and Applications*, 2003.

[8] D. Warneke and O. Kao, "Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud," *IEEE Transactions on Parallel and Distributed Systems,* vol. 6, no. 22, pp. 985-997, februar 2011.

[9] M. Shiraz, A. Gani, R. H. Khokhar and R. Buyya, "A Review on Distributed Application Processing Frameworks in Smart Mobile Devices for Mobile Cloud Computing," *IEEE Communications Surveys and Tutorials,* vol. 3, no. 15, pp. 1294-1313, 2013.

[10] Gartner, "6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015," 10 november 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3165317.

[11] K. Hwang, G. C. Fox and J. J. Dongarra, Distributed and Cloud Computing – From Parallel Processing to the Internet of Things, Morgan Kaufmann, 2012.

[12] Facebook, "Facebook Rate Limiting," [Online]. Available: https://developers.facebook.com/docs/graph-api/advanced/rate-limiting.

[13] Twitter, "API Rate Limits," [Online]. Available: https://dev.twitter.com/rest/public/rate-limiting.

[14] Gartner, "SMB Cloud Adoption Plans Present Rich Opportunities for Providers," 2015. [Online]. Available: https://www.gartner.com/doc/3009717/survey-analysis-smb-cloud-adoption.

[15] Gartner, "Worldwide Cloud Infrastructure-as-a-Service Spending to Grow 32.8 Percent in 2015," 2015. [Online]. Available: http://www.gartner.com/newsroom/id/3055225.

[16] J. Bollen, A. Pepe and H. Mao, "Modeling Public Mood and Emotion: Twitter Sentiment and Socio-Economic Phenomena," *arXiv,* vol. 0911, no. 1583, pp. 9-19, 2009.

[17] J. A. Stimson, Public Opinion in America: Moods, Cycles and Swings, Second Edition, Westview Press, 1999.

[18] R. S. Burt, "The Social Capital of Opinion Leaders," *The Annals of the American Academy of Political and Social Science,* vol. 566, no. 1, pp. 37-54, 1999.

[19] M. Cha, H. Haddadi, F. Benevenuto and K. P. Gummadi, "Measuring User Influence in Twitter: The Million Follower Fallacy," in *Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media*, 2010.

[20] E. Dubois, "The Multiple Facets of Influence: Identifying Political Influentials and Opinion Leaders on Twitter," *American Behavioral Scientist,* vol. 58, no. 10, pp. 1260-1277, september 2014.

[21] "Two-step Flow of Communication," Wikipedia, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Two-step_flow_of_communication.

[22] Z. Tufekci, "Big Questions for Social Media Big Data: Representativeness, Validity and Other Methodological Pitfalls," *arXiv,* vol. 1403, no. 7400, 2014.

[23] D. Ruths and J. Pfeffer, "Social Media for Large Studies of Behavior," *Science,* vol. 346, no. 6213, pp. 1063-1064, 2014.

[24] C. W. Tan, D. Chiu, J. Lui and D. Yau, "A Distributed Throttling Approach for Handling High Bandwidth Aggregates," *IEEE Transactions on Parallel and Distributed Systems,* vol. 18, no. 7, pp. 983-995, 2007.

[25] J. Smailović, J.Kranjc, M. Grčar, M. Žnidaršič and I. Mozetič, "Monitoring the Twitter sentiment during the Bulgarian elections," in *Data Science and Advanced Analytics (DSAA)*, Paris, 2015.

[26] G. System, "Gama System PerceptionAnalytics," Gama System, 2016. [Online]. Available: http://www.perceptionanalytics.net.